

401 – Find an illegally filmed image

Team Information

Team Name : ForensicGPT

Team Member : Eungchang Lee, Donghyun Ha, Hyunwoo Shin, Jinhyeok Son

Email Address : forensicgpt@googlegroups.com

Instructions

Description The police obtained a Pixel 3, Android 10 version smartphone with an image of illegal filming of a suspect in the illegal filming incident. The suspect's smartphone was secured with the power off, 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 Descriptor: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:	HashTab	Publisher:	Implbits Software
Version:	6.0.0		
URL:	https://implbits.com		

Name:	fbekeyfind	Publisher:	Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU)
Version:	6.0.0		
URL:	https://www.cs1.tf.fau.de/research/system-security-group/one-key-to-rule/		

Name:	aeskeyfind	Publisher:	makomk
Version:	6.0.0		
URL:	https://github.com/makomk/aeskeyfind		

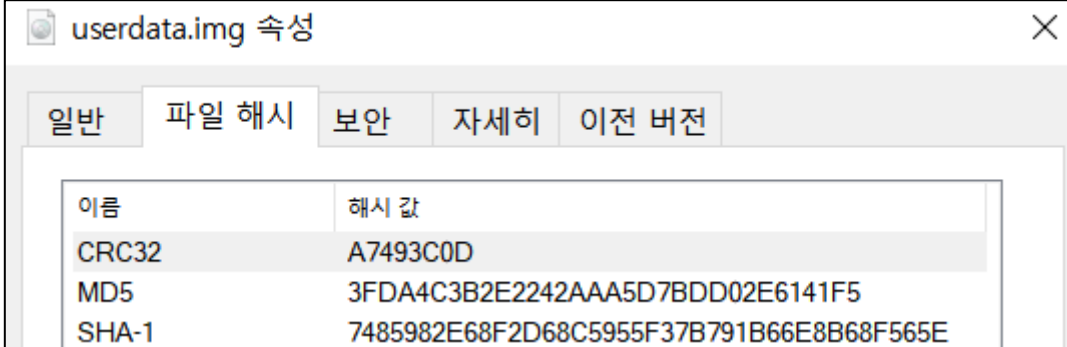
Name:	Sleuthkit-ext4-fbe	Publisher:	Tobias Groß
Version:	-		
URL:	https://fau1-gitlab.cs.fau.de/tobias.gross/sleuthkit-ext4-fbe		

Name:	pytsk	Publisher:	Tobias Groß
Version:	-		
URL:	https://fau1-gitlab.cs.fau.de/tobias.gross/pytsk3		

VM used:

Name:	Ubuntu 18.04	Publisher:	Ubuntu
Version:	5.4.0-150-generic		
URL:	https://releases.ubuntu.com/18.04/		

Step-by-step methodology:



이름	해시 값
CRC32	A7493C0D
MD5	3FDA4C3B2E2242AAA5D7BDD02E6141F5
SHA-1	7485982E68F2D68C5955F37B791B66E8B68F565E

[그림 1] userdata.img 해시 값



이름	해시 값
CRC32	0344E187
MD5	C1C33D2781B3A65BB8B0E3FAF674C5BE
SHA-1	9202325C16FF73C79FF8DFEBB443DBDC422DCA...

[그림 2] ramdump 해시 값

분석에 앞서, 주어진 두 파일에 대한 해시 값을 산출하여 MD5 해시 값이 일치함을 확인하였습니다.

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 Descriptor:MasterKey1:MasterKey2 in order, separated by colons (:), and described in one line without spaces.

```
sinsa@ubuntu:~/Desktop/one_key/fbekeyfind$ file userdata.img
userdata.img: Linux rev 1.0 ext4 filesystem data, UUID=57f8f4bc-abf4-655f-bf67-946f
c0f9f25b (needs journal recovery) (extents) (large files)
```

[그림 3] userdata.img 파일 시스템 정보 확인

주어진 용의자의 암호화된 userdata.img 는 ext4 파일 시스템 기반 파일로 확인되었습니다.

해당 문헌¹을 참고하여, FBE(File Based Encryption)이 적용된 파일 시스템에 대해 Master key 를 출력할 수 있다는 점을 알 수 있었습니다.

따라서, 다음의 명령어를 통해 Master key를 출력하였습니다.

- python3 findMasterKeys.py --part userdata.img ramdump

```
Using Cache File for Encryption Attributes
Found 1843 Files with Encryption Attributes in Image userdata.img
Found Cache File not Matching Memory Dump File - Discard Cache
Using aeskeyfind
Found 1877 AES-256-Bit Keys in Memory Dump ramdump
0 of 1843 have unexpected encryption mode
Cracking |#####| 3457468/3457468
Result for Master Key Descriptor: ec34dd2d7a20bf2e
Found Keys: a45d06abee12df0b54cbf332b936d2fe937dd7cae36074e8f42a828d509144f4, 6b65d32093a85bb36e70dfd85
40f9de5c6e0a7553f8c50e64a893f05796904db
With Hits: 553, 48, 1, 1, 1, 1, 1, 1, 1, 1
Result for Master Key Descriptor: e8aece005a118a72
Found Keys: 15584d919ff3a922b18f699dc845944a18ba1d8263666aa69269682915652d49, b9bcd156330e8b27adce3b067
619a959eae170a59e262e705f8200498dfb3e86
With Hits: 289, 218, 1, 1, 1, 1, 1, 1, 1, 1
Result for Master Key Descriptor: aff2ea76f5190ee6
Found Keys: f8a84e33e0c4efba5908da3cd1dd2349eae901aa5f9bc99b9f218434f074faef, ae96a98ef97afa52e8545f124
75640361358f38706153e225408f09e8ee57483
With Hits: 491, 27, 1, 1, 1, 1, 1, 1, 1, 1
```

[그림 4] 추출된 Master keys

이 중, 문제에서 요구한 FBE Key Descriptor 발생 빈도 기준으로 상위 2개의 FBE Master Key 는 위 그림의 Hits Count를 통해 알 수 있으며, Key는 다음 페이지의 표에 문제의 포맷과 같이 기재하였습니다.

¹ <https://www.sciencedirect.com/science/article/pii/S266628172100007X>

[표 1] 식별된 상위 2 FBE Master Keys

1	2
ec34dd2d7a20bf2e:a45d06abee12df0b54 cbf332b936d2fe937dd7cae36074e8f42a8 28d509144f4:6b65d32093a85bb36e70dfd 8540f9de5c6e0a7553f8c50e64a893f0579 6904db	aff2ea76f5190ee6:f8a84e33e0c4efba5908 da3cd1dd2349eae901aa5f9bc99b9f21843 4f074faef:ae96a98ef97afa52e8545f12475 640361358f38706153e225408f09e8ee57 483

2) Find all the file names of image files in the DCIM folder. (60 points)

```
sinsa@ubuntu:~/Desktop/one_key/fbekeyfind$ cat fbe.keys
ec34dd2d7a20bf2e:a45d06abee12df0b54cbf332b936d2fe937dd7cae36074e8f42a828d509144f4:6b65d32093a
85bb36e70dfd8540f9de5c6e0a7553f8c50e64a893f05796904db
e8aece005a118a72:15584d919ff3a922b18f699dc845944a18ba1d8263666aa69269682915652d49:b9bcd156330
e8b27adce3b067619a959eae170a59e262e705f8200498dfb3e86
aff2ea76f5190ee6:f8a84e33e0c4efba5908da3cd1dd2349eae901aa5f9bc99b9f218434f074faef:ae96a98ef97
afa52e8545f12475640361358f38706153e225408f09e8ee57483
```

[그림 5] 획득한 fbe keys

앞서 실행한 명령을 통해 fbe.keys를 획득할 수 있습니다. 해당 파일에는 KeyDescriptor:MK1:MK2 포맷을 가진 FBE Master Key 3쌍이 들어가 있는 것을 확인할 수 있습니다.

```
sinsa@ubuntu:~/Desktop/one_key/fbekeyfind$ ./fls_wrapper.sh userdata.img | head
d/d 11:
d/d 12:
+ d/d 13:
++ r/r 14:
+ d/d 15:
++ r/r 16:
+ d/d 17:
+ d/d 18:
++ r/r 19:
++ d/d 8213:
```

[그림 6] fls_wrapper.sh 사용

ext4 파일시스템 내 파일 및 디렉토리 inode number를 확인하기 위해 fls_wrapper.sh 를 사용하였습니다.

```
sinsa@ubuntu:~/Desktop/one_key/fbekeyfind$ fls -K fbe.keys userdata.img 106499
d/d $ 106500: Android
d/d $ 106498: Music
d/d $ 106502: Podcasts
d/d $ 106503: Ringtones
d/d $ 106504: Alarms
d/d $ 106505: Notifications
d/d $ 106506: Pictures
d/d $ 106507: Movies
d/d $ 106508: Download
d/d $ 106509: DCIM
```

[그림 7] key를 통해 확인한 DCIM 폴더의 inode number

fls 와 추출된 fbe.keys를 활용하여 inode number 106499에 해당하는 directory를 조회한 결과 DCIM 폴더의 inode number를 획득할 수 있었습니다.

```

sinsa@ubuntu:~/Desktop/one_key/fbekeyfind$ fls -K fbe.keys userdata.img 106509
r/r $ 106511: 52e3d54a4855ae14f1dc8460962e33791c3ad6e04e507749742c78d6944cc3_640.jpg
r/r $ 106512: 52e4d5414a5ab10ff3d8992cc12c30771037dbf852547848702a7fd0954b_640.jpg
r/r $ 106513: 52e5d7434253a814f1dc8460962e33791c3ad6e04e507440742a7ad2974bc1_640.jpg
r/r $ 106514: 55e6dd474b5baa14f1dc8460962e33791c3ad6e04e5074417c2d78d19f48c3_640.jpg
r/r $ 106515: 55e8d5424b56a514f1dc8460962e33791c3ad6e04e5074417d2e72d29e4ac3_640.jpg
r/r $ 106516: 55e8dd4a4255b10ff3d8992cc12c30771037dbf85254794e73277bd7954a_640.jpg
r/r $ 106517: black-coffee-1867753_640.jpg
r/r $ 106518: g312a1db9dd11930ce7698981e6f1353258fa24ff7faae68e148ffdd4b16d18e958e78de6751cd7df
595657572cb372c9_640.jpg
r/r $ 106519: g91d447ecfe72f9eec67075695beb60be3f06e9f341d675a76e847a2fd150139425d7ca3a1de19130
389065a4706df7a5_640.jpg
r/r $ 106520: g9eea99cb46a0b08d4521d561dd9788383f3163d335eb709f68476e600561afdcaa297e9090a7cf64
b66266cc6374d867_640.jpg
r/r $ 106521: peas-580333_640.jpg

```

[그림 8] DCIM folder 내 이미지 파일 식별

pytsk3 속 sleuthkit의 fstools에서는 fls라는 inode number에 따른 ls 명령을 확인할 수 있습니다. 암호화되어 있다면 -K 옵션과 masterkey가 들어있는 파일을 인자로 주어야 복호화 된 파일 및 디렉터리 이름들을 확인할 수 있습니다.

그래서 앞서 살펴본, fls_wrapper.sh를 통해 출력한 디렉토리 및 파일 inode number 정보에서 DCIM folder가 inode number 106509에 해당한다는 것을 알게 되었고, 획득하였던 fbe.keys를 통해 폴더 내 사진파일들의 이름을 식별할 수 있었습니다.

식별된 파일들의 이름은 다음의 표와 같습니다.

[표 2] DCIM 폴더 내 이미지 파일 이름 식별

inode num	Decrypted File Name
106511	52e3d54a4855ae14f1dc8460962e33791c3ad6e04e507749742c78d6944cc3_640.jpg
106512	52e4d5414a5ab10ff3d8992cc12c30771037dbf852547848702a7fd0954b_640.jpg
106513	52e5d7434253a814f1dc8460962e33791c3ad6e04e507440742a7ad2974bc1_640.jpg
106514	55e6dd474b5baa14f1dc8460962e33791c3ad6e04e5074417c2d78d19f48c3_640.jpg
106515	55e8d5424b56a514f1dc8460962e33791c3ad6e04e5074417d2e72d29e4ac3_640.jpg
106516	55e8dd4a4255b10ff3d8992cc12c30771037dbf85254794e73277bd7954a_640.jpg
106517	black-coffee-1867753_640.jpg
106518	g312a1db9dd11930ce7698981e6f1353258fa24ff7faae68e148ffdd4b16d18e958e78de6751cd7df595657572cb372c9_640.jpg
106519	g91d447ecfe72f9eec67075695beb60be3f06e9f341d675a76e847a2fd150139425d7ca3a1de19130389065a4706df7a5_640.jpg
106520	g9eea99cb46a0b08d4521d561dd9788383f3163d335eb709f68476e600561afdcaa297e9090a7cf64b66266cc6374d867_640.jpg
106521	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. (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.)

2번 문항에서 구한 각 이미지에 대한 FBE와 관련된 확장 속성 4 유형을 찾기 위해서는 pytsk3/sleuthkit/tools/fstools 에 있는 istat 도구 파일을 활용해볼 수 있습니다. 혹은, fbeindkey 폴더에 존재하는 istat_wrapper_enc.sh 셸 스크립트 파일을 활용해볼 수도 있습니다.

```
sinsa@ubuntu:~/Desktop/one_key/fbekeyfind$ ./istat_wrapper_enc.sh userdata.img 106511
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
```

[그림 9] 4 types of Extend Attributes 식별

위 그림은 inode number 106511에 해당하는 이미지 파일에 대한 확장 속성을 확인한 부분입니다. 해당 파일의 Content는 aes-265-xts로 암호화되어 있고, Name은 aes-256-cts로 암호화되어 있는 것을 알 수 있습니다.

이를 통해 문제 2에서 구한 11개의 이미지 파일에 대한 확장 속성을 다음 페이지의 표로 space 없이, enter 없이 한 줄로 정리하였습니다.

[표 3] 4 types of Extend Attributes related to FBE for each image file

1	52e3d54a4855ae14f1dc8460962e33791c3ad6e04e507749742c78d6944cc3_640.jpg:1:4:ec34dd2d7a20bf2e:d6698b2f318dc3e5934f9fc41453fd51
2	52e4d5414a5ab10ff3d8992cc12c30771037dbf852547848702a7fd0954b_640.jpg:1:4:ec34dd2d7a20bf2e:f74d2d27fb6de66beb50431cbadb2f49
3	52e5d7434253a814f1dc8460962e33791c3ad6e04e507440742a7ad2974bc1_640.jpg:1:4:ec34dd2d7a20bf2e:cf92299a65db6027c56ee9da77b8c3b8
4	55e6dd474b5baa14f1dc8460962e33791c3ad6e04e5074417c2d78d19f48c3_640.jpg:1:4:ec34dd2d7a20bf2e:dfad5017231c4b2d61e181d41d0939bb
5	55e8d5424b56a514f1dc8460962e33791c3ad6e04e5074417d2e72d29e4ac3_640.jpg:1:4:ec34dd2d7a20bf2e:949cec68e90588fd715096e1b1aa5d3d
6	55e8dd4a4255b10ff3d8992cc12c30771037dbf85254794e73277bd7954a_640.jpg:1:4:ec34dd2d7a20bf2e:5fcbbf00942d371e1b73ace806786afa
7	black-coffee-1867753_640.jpg:1:4:ec34dd2d7a20bf2e:51c34b6909ea3598daad02214f2af0f6
8	g312a1db9dd11930ce7698981e6f1353258fa24ff7faae68e148ffdd4b16d18e958e78de6751cd7df595657572cb372c9_640.jpg:1:4:ec34dd2d7a20bf2e:4bb0ab4e5c3ac8f00b14260418402a06
9	g91d447ecfe72f9eec67075695beb60be3f06e9f341d675a76e847a2fd150139425d7ca3a1de19130389065a4706df7a5_640.jpg:1:4:ec34dd2d7a20bf2e:904d446a5f5a7710bf87a4dcc59851e
10	g9eea99cb46a0b08d4521d561dd9788383f3163d335eb709f68476e600561afdcaa297e9090a7cf64b66266cc6374d867_640.jpg:1:4:ec34dd2d7a20bf2e:ac4589492e37397f1e26343e2d8c57d7
11	peas-580333_640.jpg:1:4:ec34dd2d7a20bf2e:2674fc48b1e313579afb5c25c80aad6c

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.)

```
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;
}
```

[그림 10] DEK 파생 함수(KDF)

위 참고 문헌에서 확인한 그림을 기반으로, 각 파일에 대한 DEK를 얻기 위해서는 KDF 함수가 필요합니다. 해당 함수를 분석해보면 deriving key를 각 file의 nonce로, source_key를 master key로 설정하여 DEK를 얻을 수 있습니다. 또한, AES_128_ECB로 encryption을 수행하여야 정상적인 DEK가 출력되는 것도 분석을 통해 알 수 있습니다.

```
from Crypto.Cipher import AES
import binascii

deriving_key_hex = "d6698b2f318dc3e5934f9fc41453fd51"
source_key_hex = "6b65d32093a85bb36e70dfd8540f9de5c6e0a7553f8c50e64a893f05796904db"
#source_key_hex = "a45d06abee12df0b54cbf332b936d2fe937dd7cae36074e8f42a828d509144f4"

deriving_key = binascii.unhexlify(deriving_key_hex)
source_key = binascii.unhexlify(source_key_hex)

cipher = AES.new(deriving_key, AES.MODE_ECB)

derived_key = cipher.encrypt(source_key)

print(binascii.hexlify(derived_key).decode())
```

[그림 11] aes-128-ecb encryption

따라서, aes-128-ecb 암호화를 통해 DEK를 획득하기 위한 python code를 작성했습니다.

이때, DCIM 폴더 내 이미지 파일에 대한 FBE descriptor가 ec34dd2d7a20bf2e였기 때문에, 해당 descriptor에 따른 Master key인

a45d06abee12df0b54cbf332b936d2fe937dd7cae36074e8f42a828d509144f4와

6b65d32093a85bb36e70dfd8540f9de5c6e0a7553f8c50e64a893f05796904db를 source_key로 사용하였습니다.

이때, MK1인

a45d06abee12df0b54cbf332b936d2fe937dd7cae36074e8f42a828d509144f4 과,

MK2인 6b65d32093a85bb36e70dfd8540f9de5c6e0a7553f8c50e64a893f05796904db 각각 파일에 대한 nonce값과 함께 암호화했을 때 나온 DEK값이 해당 list에 존재하는 것을 알 수 있습니다.

```
C: > Users > ehfeh > Desktop > 401 - Find an illegally filmed image > dump_images > dek_retry.keys
259 b7c83d30c9db1be792930a0d3082782a4a34ee060c0c5f4cf03533df3 > |e1ff6e7fefde3b5c5825| Aa .ab, * 1의 1
260 b72b877f3d01bed88b3dfe5ec2ee4d2940d6472fc3c11641f9c8e8115005e545
261 9723762d0cbaf02619ef0640e7679b679563f4c348fef286998587f37e157961
262 bd2ca92e93cd7bdbc6c67371264db5acc7b48ff91225f2252dfb51d3a997cad1
263 ce65d1067b0e99ea24969f5d68fa37479e2e5c2938d7ecd73e48e165b5255d80
264 971193b929ea4a4f07f78ca017f1bc5182e4e72d956bb2215081abd1efc16f50
265 f0fcdcdc1e3ac4b1ab1af541d4080d1fd4e27e661f0a3d4bb808eccad569bed6
266 da2062e5e5276c915d6b4c139839487d08014c937323788060326c5fa9089be3
267 4c2633b0025cf260c06f161ffa63e0374f266a25b06d945b41791b705bf1f
268 c81feb383c3e0d06958390e929590e085e7a702b3e0de1ff6e7fefde3b5c5825
```

[그림 12] MK2, nonce encryption -> DEK 획득 후 dek list에서 확인

또한, encryption 수행 후 나온 DEK를 aeskeyfind를 통해 ramdump에서 찾아낸 aes key에 존재하는지 위 그림을 통해 확인하였습니다.

다만, 문제점은 마스터 키를 재구성하기 위해 두 개의 DEK 키를 가지고는 있지만, 이 두 개의 DEK 키 중 FBE_CONTENT_DEK 를 찾는 것이 관건이었습니다.

이는, 두 가지의 근거를 통해 FBE_CONTENT_DEK로 판단하였습니다. 첫 근거는, 또 다른 참고 문헌²에서 FBE_CONTENT_DEK는 512bits의 DEK에서 하위 256 bits는 파일 내용을 암호화 하는데 사용되는 것을 알 수 있습니다.

² https://www.dbpia.co.kr/journal/articleDetail?nodeId=NODE09285647&nodeId=NODE09285647&mediaTypeCode=185005&language=ko_KR&hasTopBanner=true

```

root@ubuntu:/home/sinsa/Desktop/one_key/fbekeyfind# cat fbe2.keys
ec34dd2d7a20bf2e:a45d06abee12df0b54cbf332b936d2fe937dd7cae36074e8f42a828d509144f4:b07a8af0220eabc7c2ee454ed0
a1ca33aaf4c0be407de0e3535608509db9197
root@ubuntu:/home/sinsa/Desktop/one_key/fbekeyfind# icat -K fbe2.keys userdata.img 106511 | xxd | head
00000000: d54e c666 86f8 c0eb 0129 c8a4 50e5 cc9f  .N.f.....)..P...
00000010: 505b 95a6 8589 0ad6 c0e7 4097 9183 3109  P[.....@...1.
00000020: 732e 07ce 2acf 2ddd 0a01 4145 fc46 6fcb  s...*...AE.Fo.
00000030: 1d8f 6897 b6c0 5212 ef95 5ea9 17af 2d3f  ..h...R...^...?
00000040: bb51 0c4e 83ac e0e9 fab1 d6d7 d4e9 4dc7  .Q.N.....M.
00000050: 43cb 6131 fc01 803c 751a 52fa 0424 685d  C.a1...<u.R..$h]
00000060: 2922 3e0d a384 9360 f9ff a7fc fa7c 121e  )">....`.....|..
00000070: 79c1 5e93 f3fc 899a ff83 a5f9 26c2 ca37  y.^.....&..7
00000080: 84f2 d51b 69a8 be01 bdbd 2c0a 7d95 36bd  ....t.....,).6.
00000090: 7d90 e8b1 fe43 81fb 3009 7b0c 72d8 1c0a  }....C..0.{.r...
root@ubuntu:/home/sinsa/Desktop/one_key/fbekeyfind# fls -K fbe2.keys userdata.img 106509
r/r $ 106511: 52e3d54a4855ae14f1dc8460962e33791c3ad6e04e507749742c78d6944cc3_640.jpg
r/r $ 106512: 52e4d5414a5ab10ff3d8992cc12c30771037dbf852547848702a7fd0954b_640.jpg
r/r $ 106513: 52e5d7434253a814f1dc8460962e33791c3ad6e04e507440742a7ad2974bc1_640.jpg
r/r $ 106514: 55e6dd474b5baa14f1dc8460962e33791c3ad6e04e5074417c2d78d19f48c3_640.jpg
r/r $ 106515: 55e8d5424b56a514f1dc8460962e33791c3ad6e04e5074417d2e72d29e4ac3_640.jpg
r/r $ 106516: 55e8dd4a4255b10ff3d8992cc12c30771037dbf85254794e73277bd7954a_640.jpg
r/r $ 106517: black-coffee-1867753_640.jpg
r/r $ 106518: g312a1db9dd11930ce7698981e6f1353258fa24ff7faae68e148ffdd4b16d18e958e78de6751cd7df595657572cb3
72c9_640.jpg

```

[그림 13] MK1만 존재한다면 파일 이름만 해독

문제 오류사항이 발생하기 전에 추출했던 Master key에서 MK2부분이 현재 FBE file descriptor ec34dd2d7a20bf2e에 대한 MK2부분과 다르다는 점을 알 수 있습니다.

이때, 저장해 놓은 fbe2.keys를 가지고 fls 명령어를 통해 파일 이름을 해독하였을 때는 정상적으로 파일 이름이 복호화 되는 것을 알 수 있습니다. 다만, icat을 통해 파일 내용을 해독하였을 때는 정상적으로 파일 내용이 해독되지 않는 것을 알 수 있습니다.

```

root@ubuntu:/home/sinsa/Desktop/one_key/fbekeyfind# cat test6.keys
ec34dd2d7a20bf2e:a45d06abee12df0b54cbf332b936d2fe937dd7cae36074e8f42a828d509144f4:6b65d32093a85bb36e70dfd8540f9de5c6
e0a7553f8c50e64a893f05796904db
root@ubuntu:/home/sinsa/Desktop/one_key/fbekeyfind# icat -K test6.keys userdata.img 106511 | xxd | head
00000000: ffd8 ffe0 0010 4a46 4946 0001 0101 012c  ....JFIF.....
00000010: 012c 0000 ffe1 00d6 4578 6966 0000 4d4d  ....Exif..MM
00000020: 002a 0000 0008 0004 010f 0002 0000 0012  .*.....
00000030: 0000 003e 0110 0002 0000 000c 0000 0050  ...>.....P
00000040: 829a 0005 0000 0001 0000 005c 8769 0004  ....\..t..
00000050: 0000 0001 0000 0064 0000 0000 4e49 4b4f  ....d....NIKO
00000060: 4e20 434f 5250 4f52 4154 494f 4e00 4e49  N CORPORATION.NI
00000070: 4b4f 4e20 4435 3230 3000 0000 0001 0000  KON D5200.....
00000080: 00fa 0005 829a 0005 0000 0001 0000 00a2  ....
00000090: 829d 0005 0000 0001 0000 00aa 8827 0003  ....'..
root@ubuntu:/home/sinsa/Desktop/one_key/fbekeyfind# fls -K test6.keys userdata.img 106509
r/r $ 106511: 52e3d54a4855ae14f1dc8460962e33791c3ad6e04e507749742c78d6944cc3_640.jpg
r/r $ 106512: 52e4d5414a5ab10ff3d8992cc12c30771037dbf852547848702a7fd0954b_640.jpg
r/r $ 106513: 52e5d7434253a814f1dc8460962e33791c3ad6e04e507440742a7ad2974bc1_640.jpg
r/r $ 106514: 55e6dd474b5baa14f1dc8460962e33791c3ad6e04e5074417c2d78d19f48c3_640.jpg
r/r $ 106515: 55e8d5424b56a514f1dc8460962e33791c3ad6e04e5074417d2e72d29e4ac3_640.jpg
r/r $ 106516: 55e8dd4a4255b10ff3d8992cc12c30771037dbf85254794e73277bd7954a_640.jpg
r/r $ 106517: black-coffee-1867753_640.jpg
r/r $ 106518: g312a1db9dd11930ce7698981e6f1353258fa24ff7faae68e148ffdd4b16d18e958e78de6751cd7df595657572cb372c9_64
0.jpg
r/r $ 106519: g91d447ecfe72f9eec67075695beb60be3f06e9f341d675a76e847a2fd150139425d7ca3a1de19130389065a4706df7a5_64
0.jpg
r/r $ 106520: g9eea99cb46a0b08d4521d561dd9788383f3163d335eb709f68476e600561afdc9a297e9090a7c6f64b66266cc6374d867_64
0.jpg
r/r $ 106521: peas-580333_640.jpg

```

[그림 14] 정상적인 FBE_FILE_DESCRIPTOR:MK1:MK2를 통한 복호화

이번에는 정상적인 Master key를 추출해서 test6.keys로 저장한 다음 fls 명령과 icat 명령을 활용해보았습니다. 뒤 마스터키 MK2만 바뀌었을 뿐인데 정상적으로 파일 내용이 해독되는 것을 알 수 있습니다. 따라서, 뒤 DEK 256bit key가 파일 내용 암호화에 사용되는 키로 판단하였습니다.

aes-256-xts모드는 두 개의 key와 tweak를 사용하고, aes-256-cts는 cbc에서 cipher text stealing을 진행하는 것이기 때문에 key가 한 개만 존재하면 되기 때문이라고 유추 가능했습니다. 따라서, 문제에서 요구하는 포맷에 따라 작성한 답은 다음의 표와 같습니다.

[표 4] FILENAME:FBE_NONCE:FBE_CONTENT_DEK

1	52e3d54a4855ae14f1dc8460962e33791c3ad6e04e507749742c78d6944cc3_640.jpg:d6698b2f318dc3e5934f9fc41453fd51:c81feb383c3e0d06958390e929590e085e7a702b3e0de1ff6e7fefde3b5c5825
2	52e4d5414a5ab10ff3d8992cc12c30771037dbf852547848702a7fd0954b_640.jpg:f74d2d27fb6de66beb50431cbddb2f49:974aaa1abd4a709dac3f21093b25a1e170538a46b818fb55de9b9760c6f0d983
3	52e5d7434253a814f1dc8460962e33791c3ad6e04e507440742a7ad2974bc1_640.jpg:cf92299a65db6027c56ee9da77b8c3b8:0a20e2c2392b49820685085c9318dcab2c51722c2d2c992dae884227956ccdb9
4	55e6dd474b5baa14f1dc8460962e33791c3ad6e04e5074417c2d78d19f48c3_640.jpg:dfad5017231c4b2d61e181d41d0939bb:da2062e5e5276c915d6b4c139839487d08014c937323788060326c5fa9089be3
5	55e8d5424b56a514f1dc8460962e33791c3ad6e04e5074417d2e72d29e4ac3_640.jpg:949cec68e90588fd715096e1b1aa5d3d:971193b929ea4a4f07f78ca017f1bc5182e4e72d956bb2215081abd1efc16f50
6	55e8dd4a4255b10ff3d8992cc12c30771037dbf85254794e73277bd7954a_640.jpg:5fcbbf00942d371e1b73ace806786afa:a79b8dc1b12637452514d254f47c29c745b12cac2c9488f720c4c85aba05852d
7	black-coffee-1867753_640.jpg:51c34b6909ea3598daad02214f2af0f6:bd2ca92e93cd7bdbc6c67371264db5acc7b48ff91225f2252dfb51d3a997cad1
8	g312a1db9dd11930ce7698981e6f1353258fa24ff7faae68e148ffdd4b16d18e958e78de6751cd7df595657572cb372c9_640.jpg:4bb0ab4e5c3ac8f00b14260418402a06:b72b877f3d01bed88b3dfe5ec2ee4d2940d6472fc3c11641f9c8e81150d5e545
9	g91d447ecfe72f9eec67075695beb60be3f06e9f341d675a76e847a2fd150139425d7ca3a1de19130389065a4706df7a5_640.jpg:904d446a5f5a7710bf87a4dccb59851e:ed7d6400235f5f4789883cefd0987ac1552f30751fdb5fd23eb2e5189986974c
10	g9eea99cb46a0b08d4521d561dd9788383f3163d335eb709f68476e600561afdcaa297e9090a7cf64b66266cc6374d867_640.jpg:ac4589492e37397f1e26343e2d8c57d7:2daa2e71c00e77bbd199f2c8fa7636a736618d27fbb494d3b397db760ba2fba2
11	peas-580333_640.jpg:2674fc48b1e313579afb5c25c80aad6:c096de4d9c559b5130d799e3e04a60894cc9787b87a0942a98936b0426c52519

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.

앞서 4번 문제에서 icat 을 통해 master_key를 가지고 파일 내용 복호화가 수행되는 것을 확인 하였습니다. 따라서, icat 과 관련된 코드를 분석하며 복호화 과정을 추적해나갔습니다.

● icat.cpp

```
#include "tsk/tsk_tools_i.h"  
#include "tsk/fs/apfs_fs.h"  
#include <locale.h>  
#include "tsk/fs/tsk_ext2fs.h"
```

[그림 15] icat.cpp include header file

icat에서는 tsk_ext2fs.h를 include 하는 것을 볼 수 있습니다.

- ext2fs.c

```

        aes_xts_decrypt_buffer_block_num(buf, fs->block_size, file_key, block_number);
        bytes_to_copy = fs->block_size - byteoffset_toread;
        if (bytes_to_copy > len_remain)
            bytes_to_copy = len_remain;
        memcpy(&a_buf[len_toread - len_remain], &buf[byteoffset_toread], bytes_to_copy);
        byteoffset_toread = 0;
        len_remain -= bytes_to_copy;

        block_number++;
        if (len_remain <= 0)
            break;
    }
    free(buf);

}

// reset this in case we need to also read from the next run
byteoffset_toread = 0;
}
return (ssize_t) (len_toread - len_remain);
}

```

[그림 16] ext2fs.c 내 ext4_read_encrypted_data 함수

ext2_fs에서도 tsk_ext2fs.h를 include합니다. 또한, DCIM 내 image file에 적용된 FBE content_mode는 xts mode(1, AES 256 XTS)였기 때문에, ext4_read_encrypted_data에서 aes_xts_decrypt_buffer_block_num 함수를 호출하는 부분을 주목할 수 있습니다.

이때 해당 ext_read_encrypted_data 함수에서는 aes_xts_buffer_block_num을 호출하면서 block_number를 전달하는데, +1 씩 증가시키면서 전달하는 것을 알 수 있습니다.

이를 통해서, 특정 inode number에 저장된 파일에 대한 데이터가 block단위로 저장되어 있고, 이 block 들을 ext2fs.c에서 읽어들이어서, aes_xts_decrypt_buffer_block_num으로 넘겨 블록 단위로 xts 암호화를 수행하는 것을 확인하였습니다.

```

#include "tsk_fs_i.h"
#include "tsk_ext2fs.h"
#include "tsk/base/crc.h"
#include <stddef.h>

#include "../util/crypto_ext.h"

```

[그림 17] ext2_fs.c 소스코드 내 include header file

또한 ext2_fs.c에서는 aes_xts_buffer_block_num을 호출하기 위해 ../util/crypto_ext.h를 include하는 것을 알 수 있습니다.

- crypto_ext.c

```
extern int aes_xts_decrypt_buffer(void* buffer, const size_t length, const uint8_t* key, const uint8_t* iv) {
    EVP_CIPHER_CTX *ctx;
    int outlen;

    // debug_print_buf(iv, 16);

    if ( !(ctx = EVP_CIPHER_CTX_new()) )
        return -1;

    if(1 != EVP_DecryptInit_ex(ctx, EVP_aes_256_xts(), NULL, key, iv))
        goto errorExit;

    if(1 != EVP_DecryptUpdate(ctx, (uint8_t*) buffer, &outlen, (uint8_t*) buffer, length))
        goto errorExit;

    return outlen;

errorExit:
    EVP_CIPHER_CTX_free(ctx);
    return -1;
}

extern int aes_xts_decrypt_buffer_block_num(void* buffer, const size_t length, const uint8_t* key, const size_t iv){
    uint8_t iv_bytes[16];
    int i;
    for(i=0; i < 16; i++){
        if(i<8){
            iv_bytes[i] = ((uint8_t)(iv >> (i * 8)) & 0xFF);
        }else{
            iv_bytes[i] = 0;
        }
    }
    return aes_xts_decrypt_buffer(buffer, length, key, &iv_bytes);
}
```

[그림 18] crypto_ext.c 내 aes_xts_decrypt 관련 함수

crypto_ext.h 헤더 파일을 include하는 crypt_ext.c에서 aes_xts_decrypt_buffer_block_num과 aes_xts_decrypt_buffer를 찾을 수 있었습니다.

aes_xts_decrypt_buffer_block_num에서 받아온 block number를 iv로 저장하고, aes_xts_decrypt_buffer로 넘겨주는 것을 알 수 있습니다. 그리고 해당 함수에서는 openssl 관련 함수인 EVP 함수를 통해 Decrypt를 진행하는 것을 알 수 있습니다.

이때 주목해야 할 점은 iv를 생성하기 위한 로직입니다. 16바이트의 iv 중 상위 8바이트는 ((uint8_t)(iv >> (i * 8)) & 0xff)연산을 수행하고, 하위 8 바이트는 0으로 채우는 것을 알 수 있습니다.

- crypto.cpp

```
int aes_xts_decryptor::decrypt_block(void *buffer, size_t length,
                                     uint64_t block) noexcept {
#ifdef TSK_MULTITHREAD_LIB
    // Take decryption lock
    std::lock_guard<std::mutex> lock{_ctx_lock};
#endif

    uint8_t tweak[16]{};
    for (int i = 0; i < 8; i++) {
        tweak[i] = (block >> (i * 8)) & 0xFF;
    }

    int outlen;
    EVP_DecryptInit_ex(_ctx, nullptr, nullptr, nullptr, tweak);
    EVP_DecryptUpdate(_ctx, static_cast<uint8_t *>(buffer), &outlen,
                      static_cast<uint8_t *>(buffer), length);

    return outlen;
}
```

[그림 19] tweak 생성 로직

crypto.cpp의 aes_xts_decryptor::decrypt_block 함수에서는 앞서 살펴본 aes_xts_decrypt_buffer_block_num 함수에서 수행한 로직과 유사한 tweak 생성 로직이 존재합니다. 단지, 해당 함수에서는 tweak를 0으로 초기화 하고 상위 8바이트만 연산을 수행하는 것만 다른 것을 확인하였습니다.

이를 통해, 앞서보았던 iv_bytes를 생성하는 과정이 tweak를 생성하는 과정이라는 것을 파악하였습니다. 그리고, 그 iv_bytes를 생성하기 위해 인자로 전달받은 iv는 block_number라는 것을 확인하였습니다.

```
inode: 106512
Allocated
Group: 13
Generation Id: 2987584813
uid / gid: 1023 / 1057
mode: rrw-rw-r--
Flags: No A-Time, Compression Error, Extents,
size: 141916
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: F7 4D 2D 27 FB 6D E6 6B EB 50 43 1C BE DB 2F 49

Inode Times:
Accessed:      2023-07-07 20:25:43.000000000 (KST)
File Modified: 2023-07-07 20:25:43.000000000 (KST)
Inode Modified: 2023-07-07 20:27:54.098443709 (KST)
File Created:  2023-07-07 20:27:54.098443709 (KST)

Direct Blocks:
426534 426535 426536 426537 426538 426539 426540 426541
426542 426543 426544 426545 426546 426547 426548 426549
426550 426551 426552 426553 426554 426555 426556 426557
426558 426559 426560 426561 426562 426563 426564 426565
426566 426567 426568
```

[그림 20] istat을 통해 확인한 가장 큰 DCIM 내 image file에 대한 정보

따라서, istat을 통해 알게 된 DCIM 폴더 내 가장 큰 사이즈를 가진 file에 대한 Direct Block을 출력해보았습니다. 그리고 Direct Blocks가 file content가 저장된 block number라는 것을 알 수 있었습니다. 따라서, 해당 Blocks들을 tweak value(16 bytes)로 변환하는 과정을 진행하였습니다.

```

import struct
import binascii

# 초기 벡터들
iv_list = [
    426534, 426535, 426536, 426537, 426538, 426539,
    426540, 426541, 426542, 426543, 426544, 426545,
    426546, 426547, 426548, 426549, 426550, 426551,
    426552, 426553, 426554, 426555, 426556, 426557,
    426558, 426559, 426560, 426561, 426562, 426563,
    426564, 426565, 426566, 426567, 426568
]

for iv in iv_list:
    iv_bytes = bytearray(16)

    for i in range(16):
        if i < 8:
            iv_bytes[i] = (iv >> (i * 8)) & 0xFF
        else:
            iv_bytes[i] = 0

    iv_hex = binascii.hexlify(iv_bytes)

    print(iv_hex.decode())

```

[그림 21] block number들을 tweak로 변환하는 과정

위 과정을 통해 블록 단위의 aes 256 xts에서 사용되는 tweak value들을 16 bytes 값으로 변환 (별 다른 포맷 명시 없음)한 값들을 다음과 같이 ,로 구분하여 다음 페이지에 작성하였습니다.

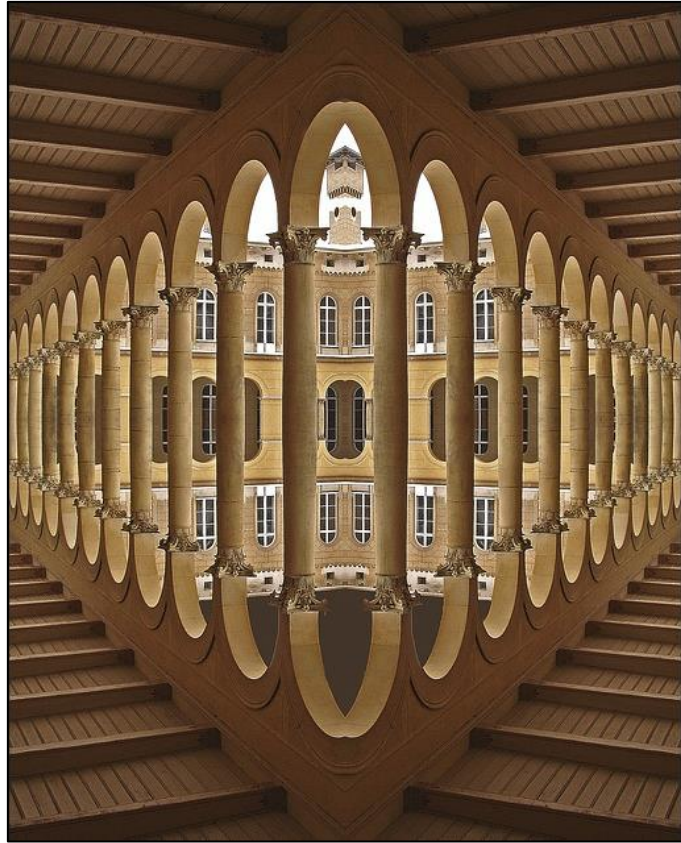
- tweak values list:

[illegible]

또한, aes-256-xts 복호화 과정을 통해 얻은 inode 106512에 해당하는 image file의 sha256 hash는 다음과 같습니다.

- FILE_CONTENT_SHA256_HASH

33a51636684e932a956d53a686e63bfd2528bf2e87ad55fdede78261e5c68ef3



[그림 22] 복호화 한 이미지 파일

inode number 106512에 해당하는 이미지는 위와 같습니다.