

251 – Find the suspect

Team Information

Team Name : ForensicGPT

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

Email Address : forensicgpt@googlegroups.com

Instructions

Description Investigators impound a MacBook believed to have been used in the crime at the crime scene. However, it is difficult to solve the case because the suspect of the crime is not specified. Analyze the evidence file to identity of the criminal.

Target	Hash (MD5)
Users.zip	56E3072B8D12D449B57E47A90BB35CAF

Questions

- 1) Find all files that appear to be related to the crime, and identify the file name and upload or download time (UTC+9) (20 points)
- 2) Identify suspect's name and email address. (80 points)
- 3) Submit the tool to decrypt the dbx. (150 points)

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).

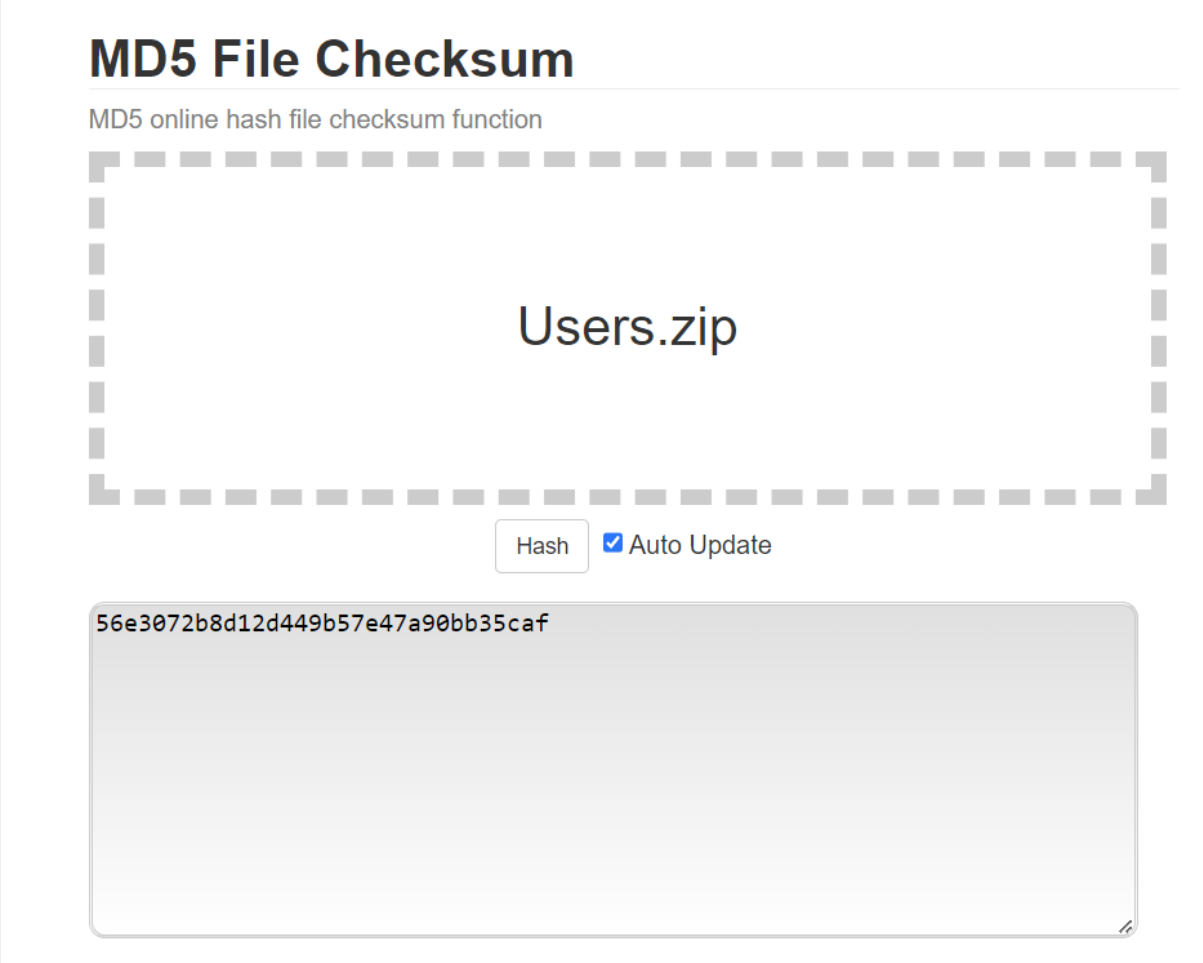
Tools used:

Name:	MD5 File checksum Online	Publisher:	-
Version:	-		
URL:	https://emn178.github.io/online-tools/md5_checksum.html		

Name:	DB Browser for SQLite	Publisher:	Patreon
Version:	3.12.2		
URL:	https://sqlitebrowser.org/		

Name:	sqlite-dbx-win64.exe	Publisher:	zena forensic
Version:	-		
URL:	https://github.com/dfirfpi/decwindbx		

Step-by-step methodology:



The screenshot shows the 'MD5 File Checksum' website. At the top, the title 'MD5 File Checksum' is displayed in a large, bold font. Below it, the subtitle 'MD5 online hash file checksum function' is visible. A large dashed rectangular box contains the text 'Users.zip'. Below this box, there are two buttons: 'Hash' and 'Auto Update' (which has a checked checkbox). At the bottom, a large light gray box displays the MD5 hash '56e3072b8d12d449b57e47a90bb35caf'.

[그림 1] 무결성 검증

먼저, 다운받은 Users.zip에 대해 무결성이 일치하는 지 확인하는 작업을 수행하였습니다.

1) Find all files that appear to be related to the crime, and identify the file name and upload or download time (UTC+9) (20 points)

Name	Date modified	Type	Size
Earlier this year			
Crashpad	2023-04-25 오후 6:00	File folder	
events	2023-04-25 오후 6:00	File folder	
instance1	2023-04-25 오후 6:00	File folder	
logs	2023-04-25 오후 6:00	File folder	
machine_storage	2023-04-25 오후 6:00	File folder	
metrics	2023-04-25 오후 6:00	File folder	
QuitReports	2023-04-25 오후 6:00	File folder	
apex.sqlite3	2023-04-25 오후 6:00	SQLITE3 File	4 KB
dropbox.pid	2023-04-25 오후 6:00	PID File	1 KB
info.json	2023-04-25 오후 6:00	JSON Source File	1 KB

[그림 2] Users.zip 내 존재하는 파일들

압축을 풀면 위 그림에서 확인가능한 드롭박스 관련 파일들이 존재한다는 사실을 알 수 있습니다.

Name	Date modified	Type	Size
Earlier this year			
avatacache.db	2023-04-25 오후 6:00	Data Base File	12 KB
config.dbx	2023-04-25 오후 6:00	DBX File	92 KB
hostkeys	2023-04-25 오후 6:00	File	1 KB
preview_cache.db	2023-04-25 오후 6:00	Data Base File	12 KB
sync_history.db	2023-04-25 오후 6:00	Data Base File	8 KB
bi_sync	2023-04-25 오후 6:00	File folder	
sync	2023-04-25 오후 6:00	File folder	

[그림 3] 범죄와 연관된 파일

업로드 혹은 다운로드 파일에 대한 정보를 담고있는 sync_history.db 파일은 Users\dfc2023\.dropbox\instance1 폴더 내에 위치하고 있다는 사실을 확인하였습니다.

테이블(T): sync_history										모든 열에서 필터링									
nt_t		event		direction		file_id		local_path		server_path		r_		timestamp					
필터		필터		필터		필터		필터		필터		...		필터					
1	file	add	download	XzV1LSzXoNQAAAAAAAAABg		/Users/dfc2023/Dropbox/입금주소(XMR).txt		3982357937-/입금주소(XMR).txt		0		1681081949							
2	file	add	download	XzV1LSzXoNQAAAAAAAAABw		/Users/dfc2023/Dropbox/1월_메스암페타민_판매_장부.xlsx		3982357937-/1월_메스암페타민_판매_장부.xlsx		0		1681081949							
3	file	add	download	XzV1LSzXoNQAAAAAAAAACA		/Users/dfc2023/Dropbox/1월_대마_판매_장부.xlsx		3982357937-/1월_대마_판매_장부.xlsx		0		1681081949							
4	file	add	download	XzV1LSzXoNQAAAAAAAAACQ		/Users/dfc2023/Dropbox/고객_정보.xlsx		3982357937-/고객_정보.xlsx		0		1681081949							
5	file	add	upload	XzV1LSzXoNQAAAAAAAAADQ		/Users/dfc2023/Dropbox/메스암페타민_샘플용.jpg		3982357937-/메스암페타민_샘플용.jpg		0		1676623739							
6	file	add	upload	XzV1LSzXoNQAAAAAAAAADA		/Users/dfc2023/Dropbox/던지기장소2.png		3982357937-/던지기장소2.png		0		1676622947							
7	file	add	upload	XzV1LSzXoNQAAAAAAAAACw		/Users/dfc2023/Dropbox/던지기장소1.png		3982357937-/던지기장소1.png		0		1676622938							

[그림 4] sync_history.db 파일 정보 확인

DB Browser for SQLite Viewer로 db파일을 확인해본 결과, 범죄와 관련된 파일들의 이름과 업로드 혹은 다운로드 시각을 다음의 표와 같이 확인할 수 있었습니다.

타입	파일명	Timestamp	시간 (UTC+9)
다운로드	입금주소(XMR).txt	1681081949	April 10, 2023 8:12:29 AM UTC+09:00
다운로드	1월_메스암페타민_판매_장부.xlsx	1681081949	April 10, 2023 8:12:29 AM UTC+09:00
다운로드	1월_대마_판매_장부.xlsx	1681081949	April 10, 2023 8:12:29 AM UTC+09:00
다운로드	고객_정보.xlsx	1681081949	April 10, 2023 8:12:29 AM UTC+09:00
업로드	메스암페타민_샘플용.jpg	1676623739	February 17, 2023 5:48:59 PM UTC+09:00
업로드	던지기장소2.png	1676622947	February 17, 2023 5:35:47 PM UTC+09:00
업로드	던지기장소1.png	1676622938	February 17, 2023 5:35:38 PM UTC+09:00

[표 1] 범죄와 관련된 파일명 및 업로드, 다운로드 시각 식별

2) Identify suspect's name and email address. (80 points)

Name	Date modified	Type	Size
✓ Earlier this year			
avatarcache.db	2023-04-25 오후 6:00	Data Base File	12 KB
config.dbx	2023-04-25 오후 6:00	DBX File	92 KB
hostkeys	2023-04-25 오후 6:00	File	1 KB
preview_cache.db	2023-04-25 오후 6:00	Data Base File	12 KB
sync_history.db	2023-04-25 오후 6:00	Data Base File	8 KB
bi_sync	2023-04-25 오후 6:00	File folder	
sync	2023-04-25 오후 6:00	File folder	

[그림 5] config.dbx 파일 확인

사용자에 대한 정보는 config.dbx에 위치하고 있습니다.

(<https://blog.digital-forensics.it/2017/04/brush-up-on-dropbox-dbx-decryption.html>)

(<http://forensicinsight.org/wp-content/uploads/2013/07/F-INSIGHT-Cloud-Storage-Forensics-Dropbox.pdf>)

위 두 URL 내 dropbox 관련 내용 분석을 통해 config.dbx는 SEE(SQLite Encryption Extension)로 암호화되어 있으며, 암호화 구조를 확인할 수 있었습니다.

Dropbox 암호화 구조에 대해 추가로 기술하자면, Dbx 암호화에는 SQLITE SEE가 사용되며, AES-128-OFB가 기본적으로 사용됩니다. SEE 암호화 시에는 DB_KEY(16 Bytes)가 사용되고, DB KEY 생성 시퀀스는 USER_KEY와 SALT(b"rc\x8c\t.\x8b\x82\xfcE(\x83\xfb_5[\xe")를 PBKDF2 해시에 입력하여 1066번 반복하여 생성됩니다. 그리고, USER_KEY는 생성된 Plain Key를 key와 iv를 통해 AES-128-CBC 암호화를 하여 hostkeys 파일에 payload로 저장되어 있음. (HMAC Validation 파일)

따라서 config.dbx를 열려면 DB_KEY가 있어야 하고, DB_KEY는 USER_KEY가 있으면 생성할 수 있기 때문에 결론적으로는 hostkeys 파일을 복호화 할 수 있어야 합니다.

<https://github.com/dnicolson/dbx-keygen-macos> URL에서 확인가능한 dbx-keygen-macos.py 코드를 살펴보면 AES_IV는 고정 값이며, KEY는 md5("ia9<dropbox 폴더의 inode 값>Xa|ui20") 이라는 것을 알 수 있습니다.

따라서, KEY 값만 brute force를 수행해보는 것으로 판단하였습니다.

(<https://github.com/dnicolson/dbx-keygen-macos>)

해당 Repository의 코드를 Python3에서 실행 가능하도록 포팅하여 문제와 동일한 환경인 맥북에서 Key를 추출해 본 결과, dropbox가 설치된 폴더의 inode를 unique id로 사용하는 방식이 유효함을 확인하였습니다.

```
eddy@hadonghyeon-ui-MacBookPro 2023-06-arrakis % python3 test.py
version : 0
[HMAL] calculated      : b'\xdd\x92\xbd~\x93z\x98Y\xd9\xca\x11U\xd0\xe4\xc3\xe4'
[HMAL] stored       : b'\xdd\x92\xbd~\x93z\x98Y\xd9\xca\x11U\xd0\xe4\xc3\xe4'
[HMAL] OK!
[PAYLOAD] ENCRYPTED : b'it4uhv\x11\xc0\xed\x80\x18\x9f;\x9a\x0b.\x89\xc8\x8f\x0cA=Dz\xb9z\x95\x9b\x89\xc8\x853\x94.\x94r(\xcc\xbe\xb6\xcf\xdb\x194K\xb6I2\xfb\x8e\xba\xc6w\xf7\x8a\xd6(\xd1l\x84e}\xc1|'
Found!
974288218528
b'{"Client": "2m+7MuZ0H1CCkuPA9spJV0LhAB3hzrN+lKtzPm5gzC0=\n"}\x04\x04\x04\x04'
Fin
```

[그림 6] inode 번호와 keygen test

이를 통해 주어진 데이터들 중 inode 값만 있다면, config.dbx를 열어 볼 수가 있습니다.

```
manager = multiprocessing.Manager()
processes = [Process(target=bruteforce_task, args=(payload, task_try_range[0], task_try_range[1])) for
               task_try_range in task_try_ranges]
```

[그림 7] multithreading logic

맥에서의 inode는 int64_t로 이는 1~9223372036854775807의 범위를 가집니다. 따라서, 순차적으로 inode를 1부터 brute forcing을 진행하기 전에, 범위가 매우 크기 때문에 멀티 스레딩으로 brute forcing을 진행하였습니다.

```

def bruteforce_task(payload: bytes, start_at: int, end_at: int):
    print(f'{start_at}~{end_at}')

    known_result_bytes = bytes('{\"Client\":', 'utf-8')
    known_result_bytes_in_ecb_mode = bytes(a ^ b for a, b in zip(known_result_bytes, aes_iv))

    payload_first_block = payload[:16]
    print('[PAYLOAD] ENCRYPTED FIRST BLOCK : ' + str(payload_first_block))

    for run_id in range(start_at, end_at):
        id2s = f'{run_id}'
        key = hashlib.md5(f'ia9{id2s}Xa|ui20'.encode()).digest()

        decryptor = AES.new(key=key, mode=AES.MODE_ECB)

        decrypted = decryptor.decrypt(bytes(payload_first_block))

        if decrypted.startswith(known_result_bytes_in_ecb_mode):
            print("=====")
            print("INODE FOUND")
            print("=====")
            print("INODE: ", run_id)
            print("DECRYPTED FIRST BLOCK IN ECB : " + str(decrypted))
            print("DECRYPTED FIRST BLOCK IN CBC : " + str(bytes(a ^ b for a, b in zip(decrypted, aes_iv))))

```

[그림 8] bruteforce logic 일부

Hostkeys의 plain data 구조는 {"Client": "~~~~~"}이라는 것을 확인하였습니다. 사용되는 암호화는 AES-128-CBC로, Overhead를 줄이기 위해 암호화된 데이터 중 첫 블록을 잘라내어, 해당 블록에 IV를 XOR 한 데이터를 처리하는 식으로 ECB로 변환한 뒤에, ECB 복호화를 진행한 데이터에서 Known Plain Data인 {"Client": 가 존재하는지 검사하는 로직을 진행하였습니다.

이후 복호화 된 Data에서 userkey를 추출하여 unversionify (HMAC 검증 후 PAYLOAD 반환 하는 과정)을 진행하였으나, 참고한 Repository의 로직과 달리 0번째 Byte인 Version Code가 0이 아니었기 때문에 이에 해당하는 HMAC_KEY를 알 수 없어 HMAC 검증이 불가능하였습니다.

그러나 구조는 동일하였기 때문에 1~17 바이트에 PAYLOAD 데이터가 위치하였고, 이를 통해 config.dbx 를 열어볼 수 있었습니다.


```

USERKEY[0] : 00fa798914ae806579c7118bb3692d69
DBKEY[0] : ef27629ffb5b22c0822577017bc3573b
Error: file is encrypted or is not a database
USERKEY[1] : fa798914ae806579c7118bb3692d6996
DBKEY[1] : 43d8f6ab723c3947aec1c0700e0185b1
USERKEY[2] : 798914ae806579c7118bb3692d6996ef
DBKEY[2] : a8bf3abf0b1209c9b6cfd3c9aba6488f
Error: file is encrypted or is not a database
USERKEY[3] : 8914ae806579c7118bb3692d6996efac
DBKEY[3] : b1c090ff16b72d781dffdd66e0ec267f
Error: file is encrypted or is not a database
USERKEY[4] : 14ae806579c7118bb3692d6996efac80
DBKEY[4] : a2c4e0f01e537eb50ffac3d02a7410d5
Error: file is encrypted or is not a database
USERKEY[5] : ae806579c7118bb3692d6996efac8012
DBKEY[5] : 8366eacb05b3d057ef2ffadad90c61a2
Error: file is encrypted or is not a database
USERKEY[6] : 806579c7118bb3692d6996efac80128e
DBKEY[6] : ff97ccc1dc07dadf511af2894aab4001

```

[그림 9] Brute force 시도

SEE에 사용된 128bit 블록 크기로 key를 잘라서 경우의 수를 모두 대입해보았을 때, 1번째에서 성공하였습니다.

config.dbx를 brute forcing을 통해 얻은 config.db 파일을 DB Browser for SQLite 로 열어보았습니다.

email	dfc2023.drugdealer@gmail.com
role	1
team	NULL
team_id	NULL
is_limited_team	0
userdisplayname	Choi danbi

확인된 용의자의 이름과 이메일 주소는 다음과 같습니다.

Suspect's name	Choi danbi
Email address	dfc2023.drugdealer@gmail.com

[표 2] 확인된 용의자 이름과 이메일 주소

3) Submit the tool to decrypt the dbx. (150 points)

- 구현 코드

```
import base64
import hmac
import hashlib
import math
import multiprocessing
import subprocess
from multiprocessing import Process
from pbkdf2 import PBKDF2
import simplejson as simplejson
from Crypto.Cipher import AES

#####
# Task Settings
#####
process_size = 1
start_at = 974288218528
end_at = 974288218530
#####
# Dropbox Constants
#####
hmac_key = b'\x8f\xf4\xf2\xbb\xad\xe9\x47\xea\x1f\xdf\x69\x6d\x80\x5b\x35\x3e'
aes_iv = b'l\x078\x014$S\x03\xffri3\x13aQ'
#####
def validateHMAC(data, hm_key, skipCheck: bool = False):
    version = data[0]
    print('[HMAC] version\t: ' + str(version))
    hm = hmac.new(hm_key, digestmod=hashlib.md5)
    digest_size = hm.digest_size
    print('[HMAC] digest_size\t: ' + str(digest_size))
    hostkeys_hmac = data[-digest_size:]
    hostkeys_payload = data[:-digest_size]
    hm.update(hostkeys_payload)
    print('[HMAC] calculated\t: ' + str(hm.digest()))
    print('[HMAC] stored\t: ' + str(hostkeys_hmac))
    if (not skipCheck) and (hm.digest() != hostkeys_hmac):
        raise Exception("hmac wrong")
    print('[HMAC] OK!')
    print(hostkeys_payload)
    payload = hostkeys_payload[1:]
    print(payload)
    return payload
def bruteforce_task(payload: bytes, start_at: int, end_at: int):
    print(f'{start_at}~{end_at}')
    known_result_bytes = bytes('{"Client":', 'utf-8')
    known_result_bytes_in_ecb_mode = bytes(a ^ b for a, b in zip(known_result_bytes,
aes_iv))
    payload_first_block = payload[:16]
    print('[PAYLOAD] ENCRYPTED FIRST BLOCK : ' + str(payload_first_block))
    for run_id in range(start_at, end_at):
        id2s = f'{run_id}'
        key = hashlib.md5(f'ia9{id2s}Xa|ui20'.encode()).digest()
        decryptor = AES.new(key=key, mode=AES.MODE_ECB)
        decrypted = decryptor.decrypt(bytes(payload_first_block))
        if decrypted.startswith(known_result_bytes_in_ecb_mode):
            print("=====")
            print("INODE FOUND")
            print("=====")
            print("INODE: ", run_id)
            print("DECRYPTED FIRST BLOCK IN ECB : " + str(decrypted))
            print("DECRYPTED FIRST BLOCK IN CBC : " + str(bytes(a ^ b for a, b in
zip(decrypted, aes_iv))))
```

```

print()
print()
print("=====")
print("FULL TEXT DECRYPT")
print("=====")
full_decryptor = AES.new(key=key, mode=AES.MODE_CBC, iv=aes_iv)
full_decrypted = full_decryptor.decrypt(bytes(payload))
pad = -(full_decrypted[-1])
full_decrypted = full_decrypted[:pad]
print("DECRYPTED : ", full_decrypted)
_dict = simplejson.loads(full_decrypted)
print("JSON : ", _dict)
userKey_payload = base64.b64decode(_dict['Client'])
print("USERKEY PAYLOAD : ", userKey_payload.hex())
print()
print()
print("=====")
print("FIND DB KEY STEP")
print("=====")
for i in range(0, len(userKey_payload)):
    print(f"STEP {i + 1}/{len(userKey_payload)}")
    userKey = (userKey_payload + userKey_payload)[i:i + 16]
    print("\t - USERKEY" + "[" + str(i) + "]" + " : " + userKey.hex())
    db_key = PBKDF2(
        passphrase=userKey,
        salt=b"\rc\x8c\t.\x8b\x82\xfcE(\x83\x9_5[\x8e",
        iterations=1066).read(16)
    print("\t - DBKEY" + "[" + str(i) + "]" + " : " + db_key.hex())
    sqliteDbxProcess = subprocess.run(
        ['./sqlite-dbx-win64.exe', '-key', (db_key.hex()), "config.dbx", ".backup
config.db"],
        stdout=subprocess.DEVNULL,
        stderr=subprocess.STDOUT
    )
    if (sqliteDbxProcess.returncode == 0):
        print("WE FOUND THAT")
        print("DECRYPTED DATABASE SAVED IN **config.db**")
        print()
        print()
        return

# Press the green button in the gutter to run the script.
if __name__ == '__main__':
    hostkeys_f = open('./hostkeys', 'rb')
    hostkeys_data = hostkeys_f.read()
    print("version : " + str(hostkeys_data[0]))
    payload = validateHMAC(hostkeys_data, hmac_key)
    print('[PAYLOAD] ENCRYPTED : ' + str(payload))
    total_size = end_at - start_at
    per_task_size = math.ceil(total_size / process_size)
    task_try_ranges = []
    for i in range(0, process_size):
        chunk_start = start_at + per_task_size * i
        chunk_end = chunk_start + per_task_size
        task_try_ranges.append([chunk_start, chunk_end])
    manager = multiprocessing.Manager()
    processes = [Process(target=bruteforce_task, args=(payload, task_try_range[0],
task_try_range[1])) for
        task_try_range in task_try_ranges]
    for process in processes:
        process.start()
    for process in processes:
        process.join()
    print(f'EXIT')

```

앞서 작성한 dbx 해독 코드는 <https://drive.google.com/file/d/16Y6QgQbeAIRTyH9WliQPJ8e0f3-Fdkub/view?usp=sharing> 에서 다운로드 받을 수 있습니다.

코드 실행 시 다음과 같이 작동되는 것을 확인할 수 있습니다.

- 코드 실행 결과

```
C:\Users\Wehfeh\Desktop\2023DFCW251 - Find the suspect>python3 main.py
version : 0
[HMAC] version : 0
[HMAC] digest_size : 16
[HMAC] calculated : b'\xdd\x92\xbd~\x93z\x98Y\xd9\xca\x11U\xd0\xe4\xc3\xe4'
[HMAC] stored : b'\xdd\x92\xbd~\x93z\x98Y\xd9\xca\x11U\xd0\xe4\xc3\xe4'
[HMAC] OK!
b'\x00it4uhv\x11\x00\xed\x80\x18\x9f;\x9a\x0b.\x89\x08\x8f\x0cA=Dz\x9b9z\x95\x9b\x89\x08\x853\x94.\x94r(\xcc\xbe\x06\xcf\xdb\x194K\xb6l2\xfb\x8e\xba\x06\x07\x8a\xd6(\xd1\x84e)\xc1|'
b'it4uhv\x11\x00\xed\x80\x18\x9f;\x9a\x0b.\x89\x08\x8f\x0cA=Dz\x9b9z\x95\x9b\x89\x08\x853\x94.\x94r(\xcc\xbe\x06\xcf\xdb\x194K\xb6l2\xfb\x8e\xba\x06\x07\x8a\xd6(\xd1\x84e)\xc1|'
[PAYLOAD] ENCRYPTED :
b'it4uhv\x11\x00\xed\x80\x18\x9f;\x9a\x0b.\x89\x08\x8f\x0cA=Dz\x9b9z\x95\x9b\x89\x08\x853\x94.\x94r(\xcc\xbe\x06\xcf\xdb\x194K\xb6l2\xfb\x8e\xba\x06\x07\x8a\xd6(\xd1\x84e)\xc1|'
974288218528~974288218530
[PAYLOAD] ENCRYPTED FIRST BLOCK : b'it4uhv\x11\x00\xed\x80\x18\x9f;\x9a\x0b.'
=====
INODE FOUND
=====
INODE: 974288218528
DECRYPTED FIRST BLOCK IN ECB : b'\x17{m}A\x1d,\xc5RK\x01~Jf'
DECRYPTED FIRST BLOCK IN CBC : b'{"Client": "2m+7'

=====
FULL TEXT DECRYPT
=====
DECRYPTED : b'{"Client": "2m+7MuZ0H1CCkuPA9spJVoLhAB3hzn+IKtzPm5gzC0=\\n"}'
```

```
JSON : {'Client': '2m+7MuZ0H1CCkuPA9spJVoLhAB3hzn+IKtzPm5gzC0=Wn'}
USERKEY                                PAYLOAD                                :
da6fbb32e6741f508292e3c0f6ca495682e1001de1ceb37e94ab733e6e60cc2d

=====
FIND DB KEY STEP
=====
STEP 1/32
    - USERKEY[0] : da6fbb32e6741f508292e3c0f6ca4956
    - DBKEY[0] : 3ca8662592ff25975b0f6a472aa012dd
STEP 2/32
    - USERKEY[1] : 6fbb32e6741f508292e3c0f6ca495682
    - DBKEY[1] : f5bfeb9d5b7905c615b2456133e6af8a
WE FOUND THAT
DECRYPTED DATABASE SAVED IN **config.db**

EXIT
```