POLITIET
KRIPOS

Uncovering Snapchat artifacts

2025-08-25 Sikkerhetsfestivalen

Ole Martin Dahl
Kripos NC3
Seksjonen for elektroniske spor / Avsnitt for metodeutvikling

https://www.politiet.no/kripos-espor

POLITIET
KRIPOS

Topics

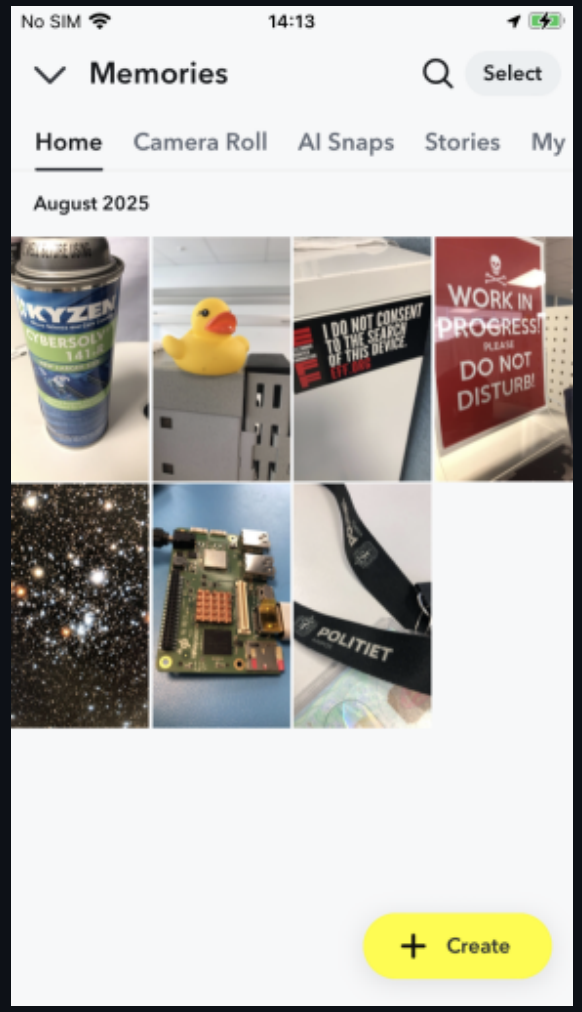
- Tools and techniques
- Some Snapchat artifacts:
  - My Eyes Only (MEO) PIN/password
  - Recovery of deleted Memories

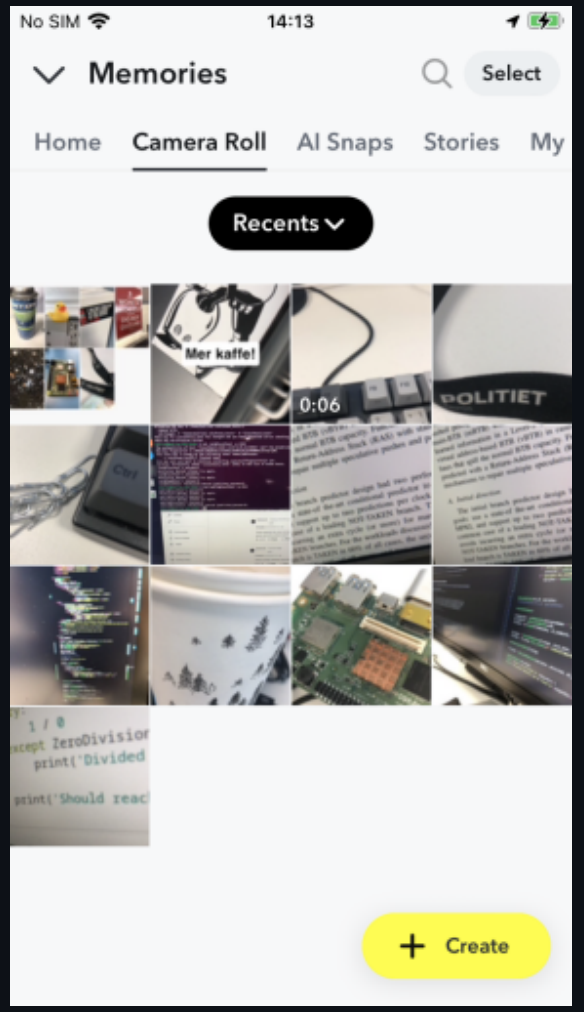
= Snapchat
= iOS
= Android

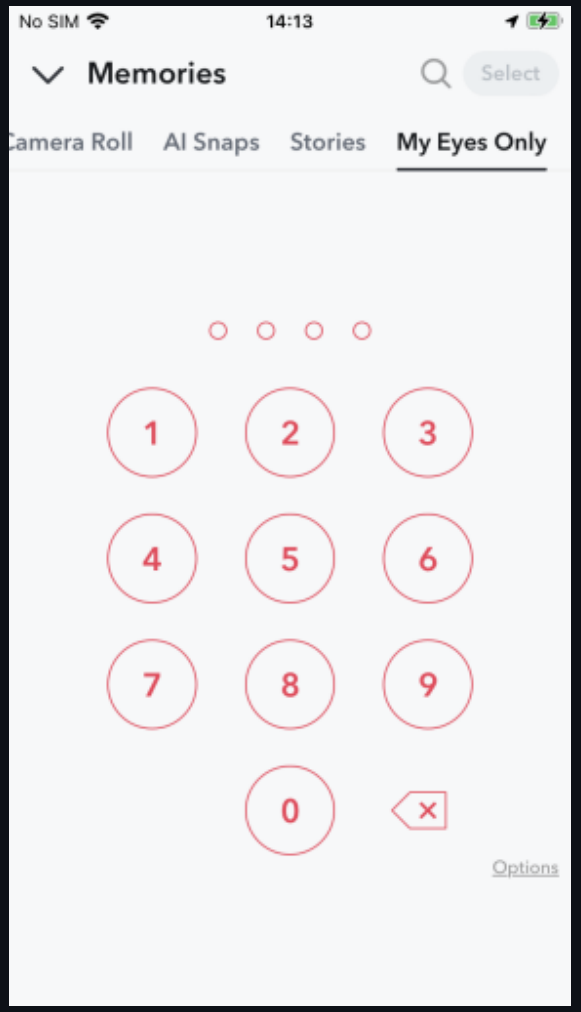
POLITIET
KRIPOS

Memories?

Pictures and videos stored in Snapchat. Both in the cloud and locally.







POLITIET
KRIPOS

Motivation

- Popularity. More than 900 million monthly users.
- and (and web/desktop).
- A moving target with varying support in commercial products.
- Very often a piece of the puzzle in criminal investigations.

Sources

- Magnet Forensics blogs on SnapChat iOS keychain usage
- Miscellaneous working and non-working scripts on github.com/gitlab.com.
  - E.g. https://github.com/DFIR-HBG
- hot3eed analysis of code obfuscation techniques in Snapchat https://hot3eed.github.io/

POLITIET
KRIPOS

Challenges with 👻

- Rapidly updated with new functionality.
- Big differences between and .
- The most interesting functionality happens in native code.
- Snapchat is challenging to reverse. E.g:
  - In 2017 Snapchat bought the company Strong.Code. This company made a effective code obfuscator using LLVM.
    - Snapchat has implemented techniques like automatic code flow changes, dead code, most library calls are done dynamically, symbols are stripped, loops are flattened and things like Mixed Boolean Arithmetic (MBA) is use to obfuscate calculations

mashable.com/article/snapchat-security-strong-codes-copying-facebook

Snap hires team to protect it from Zuckerberg's army

Strong.Codes, a Swiss startup, joined Snap this year.

By Kerry Flynn on July 21, 2017

POLITIET
KRIPOS

Tools for analysis

Root on a and a test device:

- palera1n  and Magisk 

Disassembly and decompilation with:

- Ida Pro  or Ghidra 

    - Snapchat is huge!

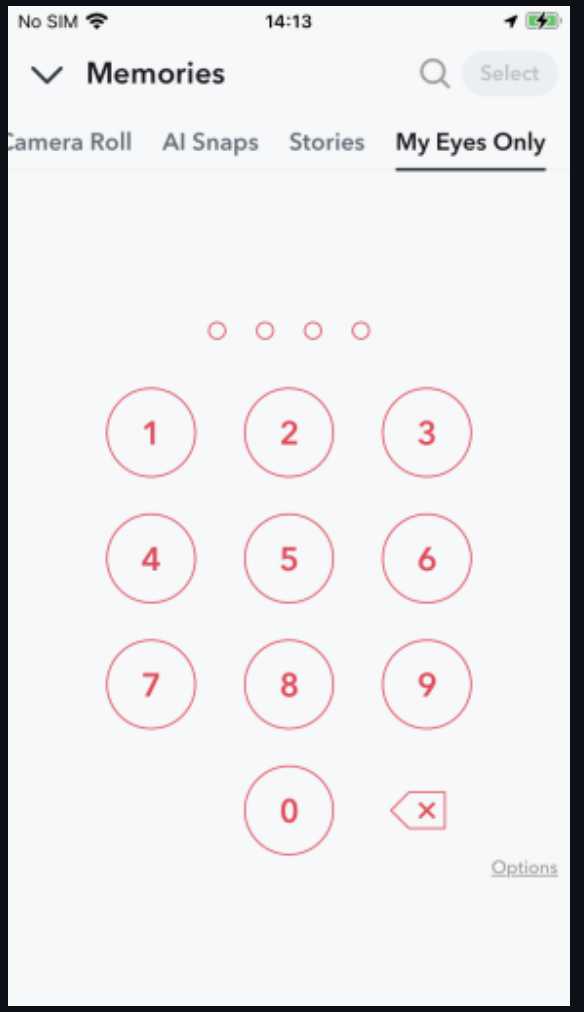
Dynamic instrumentation

- Frida.re FRIDA

POLITIET
KRIPOS

MEO pin/password

- Stored differently on and .
- If you transfer your account between and :
  - Must be online the first time you decrypt your MEO memories. The password (hash) must be stored online..



POLITIET
KRIPOS

Snapchat memories on .

Encryption and key material

An encrypted database found here:

```
/var/mobile/Containers/Data/Application/<UUID>/Documents/gallery_encrypted_db/<N>/<hash>/gallery.{encrypteddb.encrypteddb-shm,encrypteddb-wal}
```

A cleartext database found here:

```
/var/mobile/Containers/Data/Application/<UUID>/Documents/gallery_data_object/<N>/<hash>/scdb-27.{sqlite3,sqlite3-shm,sqlite3-wal}
```

Several keychain elements. E.g dumping with Frida/Objection (or commercial solutions):

```
$ objection --gadget="Snapchat" explore
com.toyopagroup.picaboo on (iPhone: 16.7.5) [net] # ios keychain dump --json snapchat_keychain.json
...
Dumped keychain to: snapchat_keychain.json
```

POLITIET
KRIPOS

memories keychain

Keychain key	Value
egocipher.key.avoidkeyderivation	Database encryption key `gallery.encrypteddb`
com.snapchat.keyservice.persistedkey	Snapchat MEO

POLITIET
KRIPOS

MEO pin/password

`com.snapchat.keyservice.persistedkey`:


```
{
        "access_control": "None",
        "accessible_attribute": "kSecAttrAccessibleWhenUnlockedThisDeviceOnly",
        "account": "com.snapchat.keyservice.persistedkey",
        "create_date": "2023-09-05 10:44:25 +0000",
        "dataHex": "62706c6973743030d4010203040506070a5824766572736696f6e592461726368697665725420.........",
        "entitlement_group": "3MY7A92V5W.com.toyopagroup.picaboo",
        "generic": "com.snapchat.keyservice.persistedkey",
        "item_class": "kSecClassGenericPassword",
        "modification_date": "2023-09-05 10:44:25 +0000",
        "service": "com.toyopagroup.picaboo",
    },
```


`"62706c6973743030d4010203040506070a5824766572736696f6e592461726368697665725420.........."`

is a NSKeyedArchiver plist object.

persistedkey NSKeyedArchiver plist

Inside the `com.snapchat.keyservice.persistedkey` NSKeyedArchiver plisten we find [Magnet forenics blog]:

Key	Value
masterKey	AES CBC (KEK)
initializationVector	AES IV
userId	A UUID
keyTag	A tag/string
passphrase	Some ciphertext of the MEO pin/password

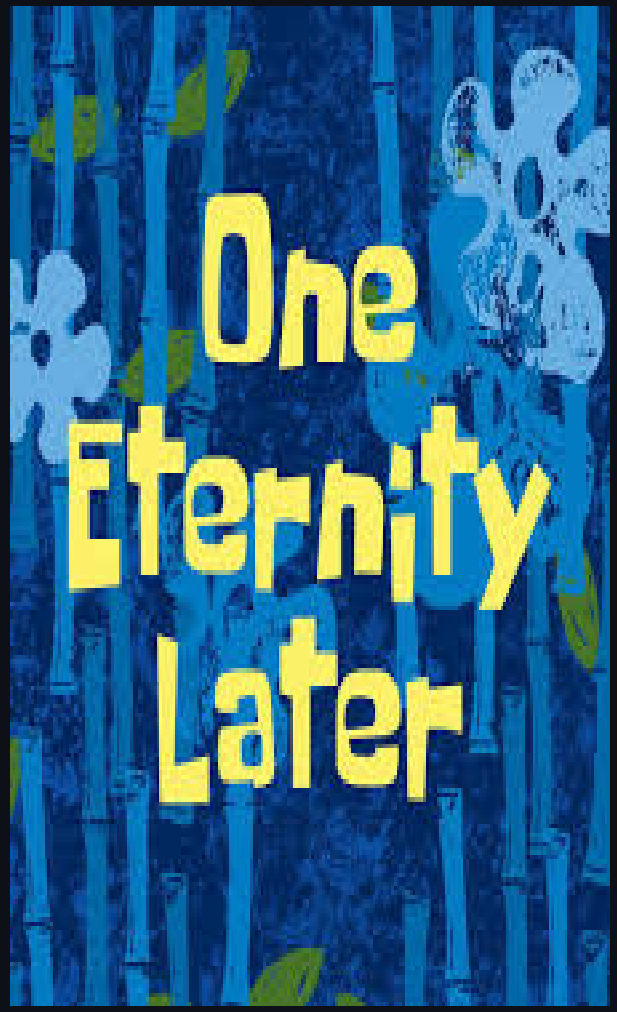
POLITIET
KRIPOS

Decompiling Snapchat

```
$ file Snapchat
Snapchat: Mach-O 64-bit arm64 executable,
flags:<NOUNDEFS|DYLDLINK|TWOLEVEL|BINDS_TO_WEAK|PIE|HAS_TLV_DESCRIPTORS>

$ ls -lah Snapchat
-rwxr-xr-x. 1 user user 225M May  5 11:24 Snapchat
```

```
$ file Snapchat.i64
Snapchat.i64: IDA (Interactive Disassembler) database

ls -lah Snapchat.i64
-rw-r--r--. 1 user user 4.2G May 15 07:16 Snapchat.i64
```

```
idaapi.load_and_run_plugin("objc", 1) //objc classes and method calls (e.g. objc_msgSend)
idaapi.load_and_run_plugin("objc", 5) //NSConcreteStackBlocks
                                      //block_layout structure applied to the function's stack frame.
idaapi.load_and_run_plugin("objc", 4) //NSConcreteGlobalBlock (globals)
```

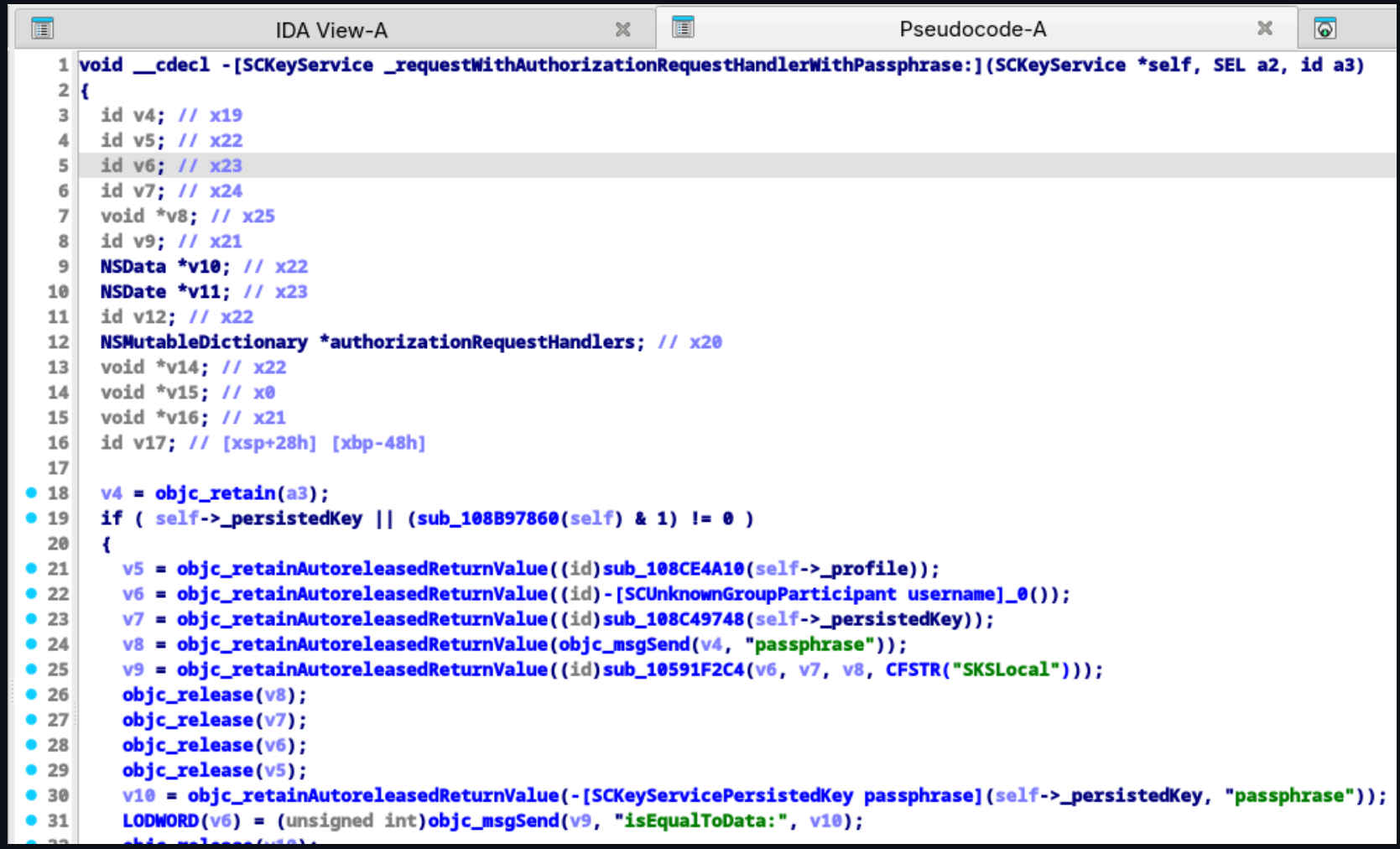


POLITIET
KRIPOS

MEO unlock

- IDA Pro Objective-C Analysis.
- Some nice symbols in Objective-C code.

E.g. `SCKeyService` class. Hooking the methods with Frida makes life much easier..



POLITIET
KRIPOS

Hooking the `_requestWithAuthorizationRequestHandlerWithPassphrase:` objective C method with FRIDA:

```
var base_addr = Module.findBaseAddress('Snapchat');
function hook__requestWithAuthorizationRequestHandlerWithPassphrase() {
  try
  {
      let methodName = "- _requestWithAuthorizationRequestHandlerWithPassphrase:";
      let address = ObjC.classes["SCKeyService"][methodName].implementation;
      console.log(`Address to ${methodName} is ${address} (Ida_addr: 0x${(address-base_addr).toString(16)})`);
      Interceptor.attach(address, {
        onEnter: function(args) {
          let a = ObjC.Object(args[0]);
          console.log(`SCKeyService _requestWithAuthorizationRequestHandlerWithPassphrase:(${a.toString()}, `);
          console.log(`\t ${ObjC.Object(args[2]).toString()},`);
        }
      });
  }
  catch(err)
...
}
hook__requestWithAuthorizationRequestHandlerWithPassphrase()
```

POLITIET
KRIPOS

`_requestWithAuthorizationRequestHandlerWithPassphrase` called with Frida Hook

```
com.toyopagroup.picaboo on (iPhone: 16.7.5) [net] # import snapchat_unlock_meo_hook.js
Snapchat base address is 0x102920000 (pid: 414})
Address to - requestAuthorizationWithPassphrase:queue:completionHandler: is 0x1081ab4e8 (Ida_addr: 0x588b4e8)
Address to - _requestWithAuthorizationRequestHandlerWithPassphrase: is 0x1081ab880 (Ida_addr: 0x588b880)
Address to scrypt/sub_4424429676 is 0x10a49706c (Ida_addr: 0x107b7706c)

# When I then enter the pin (1234) the hook triggers:

com.toyopagroup.picaboo on (iPhone: 16.7.5) [net] #
SCGalleryPrivateGalleryManager requestAuthorizationWithPassphrase:queue:completionHandler:
(<SCKeyService: 0x28360dc20>, 1234,...........
```

POLITIET
KRIPOS

Statically reversing functions called by `_requestWithAuthorizationRequestHandlerWithPassphrase:` we find encryption functions (obfuscated with no symbols).

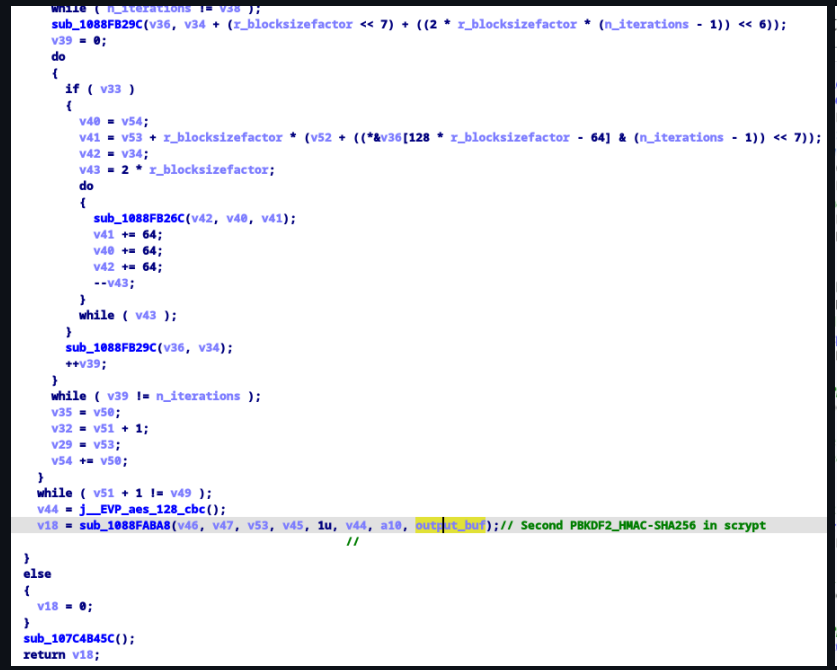
Functions with no symbols can be instrumented by calculating the functions address from the randomized base of the application.

E.g. a hook for one of the recognized PBKDF2 functions with HMAC-SHA256 as pseudo rand func:

```
var base_addr = Module.findBaseAddress('Snapchat')

function hook_sub_100ABB574(){
  try
    {
        let address = base_addr.add(0x100ABB574-0x100000000);
        console.log(`Address to sub_100ABB574 (PBKDF2WithHmacSHA256) is ${address} (Address in ida: 0x${((address-base_addr)+0x100000000).toString(16)})`);
        var outBuf;

        Interceptor.attach(address, {
          onEnter: function(args) {
            //let a = ObjC.Object(args[2]);
            console.log(`(PBKDF2WithHmacSHA256) sub_100ABB574(\n\t pass=${args[0].readCString()},`);
            console.log(`\t passLen=${args[1]} (len cleartext (${parseInt(args[1],16)}))`);
            console.log(`\t salt=${args[2]} `);
            console.log(`\t saltLen=${args[3]} (${parseInt(args[3],16)})`);
            console.log(`\t iter=${args[4]} (${parseInt(args[4],16)})`);
            console.log(`\t v14=${args[5]} `);
            console.log(`\t keyLen=${args[6]} (${parseInt(args[6],16)})`);
            console.log(`\t outBuf=${args[7]} )`);
```

POLITIET
KRIPOS

The use of two such PBKDF2 functions in one function that does something with the PIN/Password looks like the pseudo code for `scrypt` [Wikipedia scrypt article].



```
while ( n_iterations != v38 );
sub_1088FB29C(v36, v34 + (r_blocksizefactor << 7) + ((2 * r_blocksizefactor * (n_iterations - 1)) << 6));
v39 = 0;
do
{
  if ( v33 )
  {
    v40 = v54;
    v41 = v53 + r_blocksizefactor * (v52 + ((*&v36[128 * r_blocksizefactor - 64] & (n_iterations - 1)) << 7));
    v42 = v34;
    v43 = 2 * r_blocksizefactor;
    do
    {
      sub_1088FB26C(v42, v40, v41);
      v41 += 64;
      v40 += 64;
      v42 += 64;
      --v43;
    }
    while ( v43 );
  }
  sub_1088FB29C(v36, v34);
  ++v39;
}
while ( v39 != n_iterations );
v35 = v50;
v32 = v51 + 1;
v29 = v53;
v54 += v50;
}
while ( v51 + 1 != v49 );
v44 = j__EVP_aes_128_cbc();
v18 = sub_1088FABA8(v46, v47, v53, v45, 1u, v44, a10, output_buf);// Second PBKDF2_HMAC-SHA256 in scrypt
                                                  //
}
else
{
  v18 = 0;
}
sub_107C4B45C();
return v18;
```

hm [edit]

```
on scrypt
uts: This algorithm includes the following parameters:
Passphrase:                 Bytes    string of characters to be hashed
Salt:                       Bytes    string of random characters that modifies the hash to protect against
 table attacks
CostFactor (N):             Integer  CPU/memory cost parameter - Must be a power of 2 (e.g. 1024)
BlockSizeFactor (r):        Integer  blocksize parameter, which fine-tunes sequential memory read size and
ance. (8 is commonly used)
ParallelizationFactor (p):  Integer  Parallelization parameter. (1 .. 2^32-1 * hLen/MFlen)
DesiredKeyLen (dkLen):      Integer  Desired key length in bytes
                                     (Intended output length in octets of the derived key;
                                      a positive integer satisfying dkLen ≤ (2^32- 1) * hLen.)
hLen:                       Integer  The length in octets of the hash function (32 for SHA256).
MFlen:                      Integer  The length in octets of the output of the mixing function (SMix below)
 as r * 128 in RFC7914.
put:
DerivedKey:                 Bytes    array of bytes, DesiredKeyLen long

p 1. Generate expensive salt
ckSize ← 128*BlockSizeFactor  // Length (in bytes) of the SMix mixing function output (e.g. 128*8 = 1024 b

 PBKDF2 to generate initial 128*BlockSizeFactor*p bytes of data (e.g. 128*8*3 = 3072 bytes)
at the result as an array of p elements, each entry being blocksize bytes (e.g. 3 elements, each 1024 byte
...B[p-1] ← PBKDF2_HMAC-SHA256(Passphrase, Salt, 1, blockSize*ParallelizationFactor)

 each block in B Costfactor times using ROMix function (each block can be mixed in parallel)
 i ← 0 to p-1 do
B_i ← ROMix(B_i, CostFactor)

 the elements of B is our new "expensive" salt
ensiveSalt ← B_0‖B_1‖B_2‖ ... ‖B_p-1  // where ‖ is concatenation

p 2. Use PBKDF2 to generate the desired number of bytes, but using the expensive salt we just generated
urn PBKDF2_HMAC-SHA256(Passphrase, expensiveSalt, 1, DesiredKeyLen);
```

Hooking this function with Frida we get the primitives we need to attack the scrypt cipher text offline:

```
com.toyopagroup.picaboo on (iPhone: 16.5.1) [usb] # import snapchat_unlock_meo_hook.js
Snapchat base address is 0x100148000 (pid: 313})
Address to sub_100ABB7D4 is 0x100c037d4 (Address in ida: 0x100abb7d4)
com.toyopagroup.picaboo on (iPhone: 16.5.1) [usb] #
(scrypt?) sub_100ABB7D4(
        pass=32d46086-2da6-4e9a-9872-5781ef4a2170|1234|PyQI0imJlHq89KPW,
        passLen=0x3a (len cleartext (58))
        salt=SKSLocal
        saltLen=0x8 (8)
        a5=0x1000 (4096) (N iterations)
        a6=0x4 (4) (r block size?)
        a7=0x1 (1) (p parallelism factor?, usually 1)
        a8=0x0 (0) arg_0=0x0
        resultBuffer=0x281325ba0
```

POLITIET
KRIPOS

MEO hash on

```
cleartext_parts = objc_retainAutoreleasedReturnValue(snap_count(&OBJC_CLASS___NSArray));
cleartext_pwd = objc_retainAutoreleasedReturnValue(objc_msgSend(cleartext_parts, "componentsJoinedByString:", CFSTR("|")));
```

-> `passphrase` in `com.snapchat.keyservice.persistedkey` is the `scrypt` encrypted ciphertext of

```
userId|pin/passord|keyTag
```

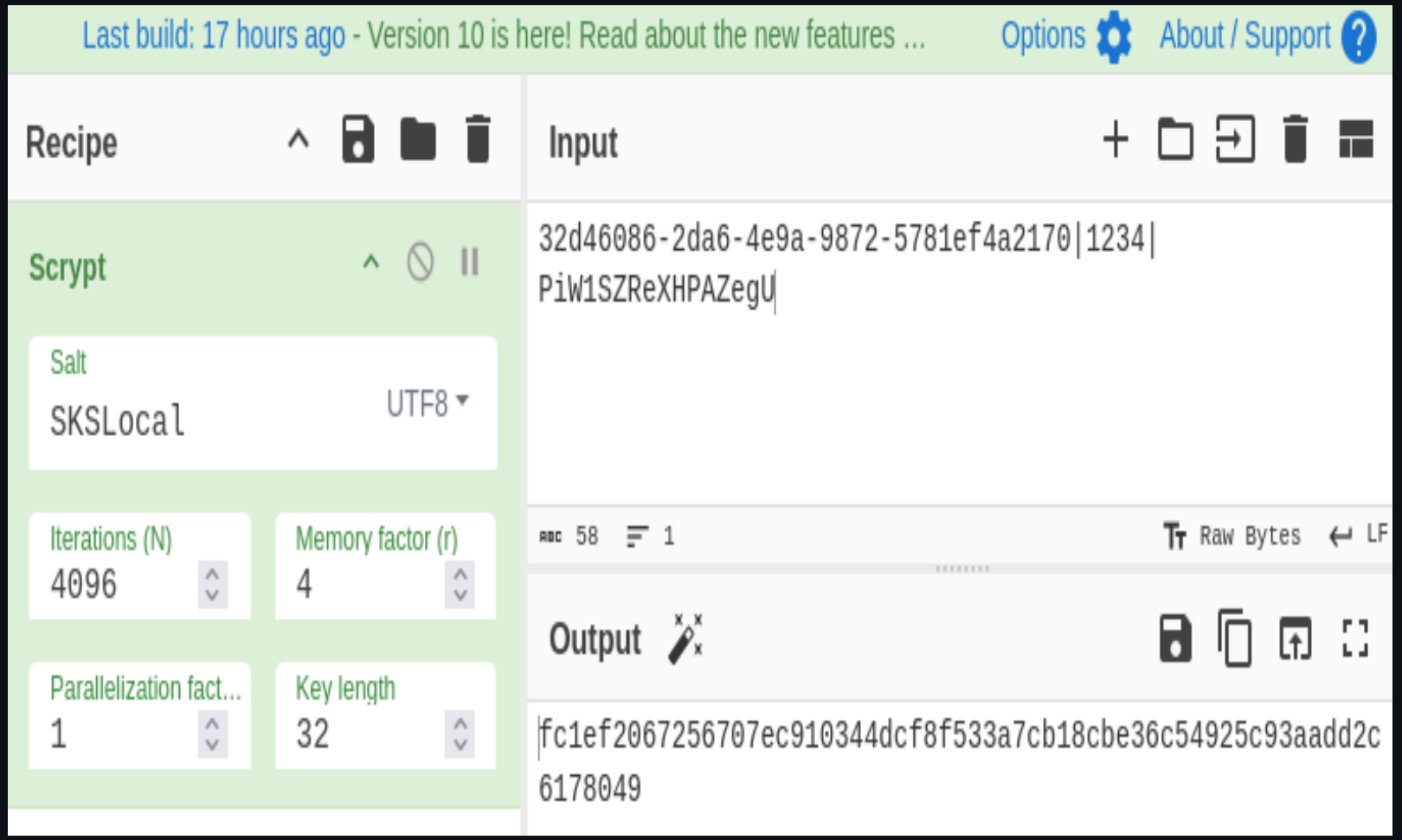
E.g.

```
32d46086-2da6-4e9a-9872-5781ef4a2170|9999|PiW1SZReXHPAZegU
```

This is input to the `scrypt` KDF using 4096 iterations (N), has a memory factor of 4 (r), parallelization factor of 1 (p), key length of 32, and hard coded `SKSLocal` string as salt.

POLITIET
KRIPOS

CyberChef POC



POLITIET
KRIPOS

Brute forcing MEO passhrase on

PoC using scrypt in python:

```
$ $ python3 snapchat-meo-brute.py --keychain snapchat_keychain.json
08-Sep-23 14:18:00 - keychain userId:
08-Sep-23 14:18:00 -    32d46086-2da6-4e9a-9872-5781ef4a2170 (36 bytes)
08-Sep-23 14:18:00 - keychain keyTag:
08-Sep-23 14:18:00 -    PiW1SZReXHPAZegU (16 bytes)
08-Sep-23 14:18:00 - keychain passphrase:
08-Sep-23 14:18:00 -    b'fc1ef2067256707ec910344dcf8f533a7cb18cbe36c54925c93aadd2c6178049' (32 bytes)
08-Sep-23 14:18:00 - keychain masterKey:
08-Sep-23 14:18:00 -    b'd77e19f45daebba9f0dc32a01b60f8abb9c9c61350a3c6c4df946b254d82be2c' (32 bytes)
08-Sep-23 14:18:00 - keychain initializationVector:
08-Sep-23 14:18:00 -    b'4688d865fc90a194f5cc6ef2c75abcbd' (16 bytes)
08-Sep-23 14:18:00 - snapchat MEO brute force starting!
08-Sep-23 14:18:05 - THE PIN CODE IS: 1234
```

Or with a hashcat `scrypt` kernel (if a password is used).

POLITIET
KRIPOS

Snapchat MEO hash on 🤖

A cleartext database here:

```
/data/data/com.snapchat.android/databases/memories.{db,db-shm,db-wal}
```

POLITIET
KRIPOS

MEO pin/password on 🤖

`memories.db`:

```
$ sqlite3 memories.db
sqlite> select * from memories_meo_confidential;
dummy|$2a$06$OymDnyM7uIvmeJVKGcFzsOKnDVMPdVP15iK/9cIxKwAz13l/HUNEe|134Z9F2uu6nw3DKgG2D4q7nJxhNQo8bE35RrJU2Cviw=
|RojYZfyQoZT1zG7yx1q8vQ==
```

Hash: `$2a$06$OymDnyM7uIvmeJVKGcFzsOKnDVMPdVP15iK/9cIxKwAz13l/HUNEe`:

- `$2a$` -> `bcrypt`
- `$06$` -> input cost, i.e. 2^6 rounds
- `OymDnyM7uIvmeJVKGcFzs` salt
- `OKnDVMPdVP15iK/9cIxKwAz13l/HUNEe` hash/ciphertext

POLITIET
KRIPOS

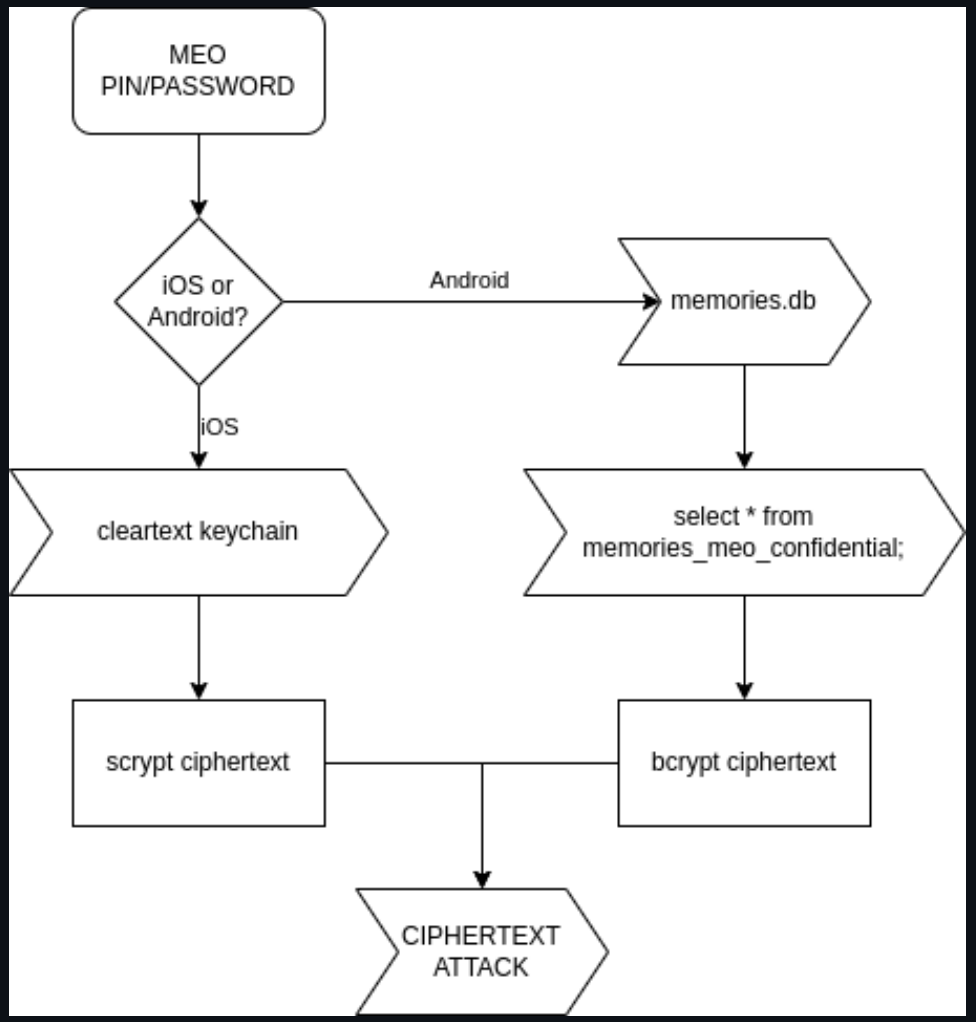
Brute forcing MEO hash on

PoC:

```
$ python3 snapchat-meo-brute.py --database memories.db
07-Sep-23 16:43:48 - Read the hash $2a$06$B2eiEYcXmX8W98TqXjMEwOfO7IiCXK/Il0wqfQcT11wQSF9JxLtlG from memories.db
07-Sep-23 16:43:48 - snapchat MEO brute force starting!
07-Sep-23 16:43:48 - THE PIN CODE IS: 1111
07-Sep-23 16:43:55 - snapchat MEO brute force done!
```

Or with a hashcat `bcrypt` kernel.

POLITIET
KRIPOS



POLITIET
KRIPOS

Download and decrypting memories on

POLITIET
KRIPOS

memories keychain "egocipher"


```
{
    "access_control": "None",
    "accessible_attribute": "kSecAttrAccessibleWhenUnlockedThisDeviceOnly",
    "account": "egocipher.key.avoidkeyderivation",
    "create_date": "2023-05-04 11:31:54 +0000",
    "dataHex": "9c66fbb7d9a9565b545b8ed87fb327a97582e9bf052acf0158246722644d5ab8",
    "entitlement_group": "3MY7A92V5W.com.toyopagroup.picaboo",
    "generic": "egocipher.key.avoidkeyderivation",
    "item_class": "kSecClassGenericPassword",
    "modification_date": "2023-05-04 11:31:54 +0000",
    "service": "com.toyopagroup.picaboo",
  },
```


From reversing and hooking we see that "egocipher" is a AES key that uses SQLCipher to encrypt and decrypt the gallery.encrytpeddb.

POLITIET
KRIPOS

memories egocipher

Now we can decrypt and make a clear text copy of `gallery.encrypteddb` like this (chiper settings hardcoded in the Snapchat Mach-O file):

```
$ sqlcipher gallery.encrypteddb
sqlite> PRAGMA key="x'242927b1d7424f5c786cc323f7034ba7c158a6fa9c71255445e16b054ebfd8a9'";
ok
sqlite> PRAGMA cipher_page_size=1024;
sqlite> PRAGMA kdf_iter=64000;
sqlite> PRAGMA cipher_hmac_algorithm=HMAC_SHA1;
sqlite> PRAGMA cipher_kdf_algorithm=PBKDF2_HMAC_SHA1;
sqlite> .recover
sqlite> .output gallery.decrypteddb.recovered
sqlite> .dump
sqlite> .exit
$ cat gallery.encrypteddb.recovered | sqlite3 gallery.decrypteddb
```

POLITIET
KRIPOS

memories egocipher

Inside of `gallery.encrypteddb` there is a key and IV pair for every snap_id.

If the memory is a My Eyes Only memory it has a boolean, `encrypted`, that says that it's key and IV is encrypted.

```
sqlite> SELECT snap_id,hex(key),hex(iv) FROM snap_key_iv WHERE encrypted=1 LIMIT 2;
ca8a4da7-bd28-3952-17c7-30bd959d65f9|
        D98EA3A176777A11B992415365763C24D02326637ED77919CCE67F588F0ED3544E024D9E0815741FC167F4E1248CC224|
        EAF68B1165DA0CE15D5CBA72C2974F5E46366E04FA2D4D8A13C4E36093854926
60A5C5A3-0EBB-4DC8-9DB9-A150F94BF4D5|
        E1849B5A3E8E846BF69F9C7A9B2129DDC93D84310EB678ED00A4C52C4D28AD5E0425948FCD4D02AEB0E74A5554283264|
        24C18EEA70DC9934B54899C19A6FD19B2D36131520BB262782C83F060C5EAD30
```

Every MEO key and IV is encrypted with the `masterkey` and `initializationVector` that we found in the keychain object `com.snapchat.keyservice.persistedkey` (which was a NSKeyedArchiver plist)

POLITIET
KRIPOS

Memories the snapchat server 👻

In the database `scdb-27.sqlite3` we find a link to every encrypted memory stored in the cloud. These we can decrypt with the keys we previously found for each `snap_id` that matches `zmediaid` or `zsnapid` (2 x AES CBC for MEO)

```
$ sqlite3 scdb-27.sqlite3
sqlite> SELECT zmediadownloadurl FROM zgallerysnap WHERE zmediaid == '60A5C5A3-0EBB-4DC8-9DB9-A150F94BF4D5' OR znapid == '60A5C5A3-0EBB-4DC8-9DB9-A150F94BF4D5';

https://cf-st.sc-cdn.net/h/fjAGkc2oIJEnAvVKL3t84?bo=Eg0aABoAMgEISAJQHGAB&uc=28
```

Get the encrypted memory from the ☁️ ☀️ ☁️

```
$ curl -o 60A5C5A3-0EBB-4DC8-9DB9-A150F94BF4D5 https://cf-st.sc-cdn.net/h/fjAGkc2oIJEnAvVKL3t84?bo=Eg0aABoAMgEISAJQHGAB&uc=28
$ file 60A5C5A3-0EBB-4DC8-9DB9-A150F94BF4D5
60A5C5A3-0EBB-4DC8-9DB9-A150F94BF4D5: data
```

POLITIET
KRIPOS

Get encrypted key material

Get the encrypted key and iv from gallery.encrypteddb

```
sqlite> SELECT snap_id,hex(key),hex(iv) FROM snap_key_iv WHERE snap_id = '60A5C5A3-0EBB-4DC8-9DB9-A150F94BF4D5';
60A5C5A3-0EBB-4DC8-9DB9-A150F94BF4D5|
E1849B5A3E8E846BF69F9C7A9B2129DDC93D84310EB678ED00A4C52C4D28AD5E0425948FCD4D02AEB0E74A5554283264|
24C18EEA70DC9934B54899C19A6FD19B2D36131520BB262782C83F060C5EAD30
```

Encrypted key:

`E1849B5A3E8E846BF69F9C7A9B2129DDC93D84310EB678ED00A4C52C4D28AD5E0425948FCD4D02AEB0E74A5554283264`

Encrypted iv:

`24C18EEA70DC9934B54899C19A6FD19B2D36131520BB262782C83F060C5EAD30`

POLITIET
KRIPOS

More key material

From the keystore `com.snapchat.keyservice.persistedkey` we have the NSKeyedArchive:

`62706c6973743030d4010203040506070a58247665727369 6f6e59246......`

```
$ pip install NSKeyedUnArchiver
$ ipython
In [1]: persistedkey = bytes.fromhex('62706c6973743030d4010203040506070a582476657273696f6e5924....')
In [2]: NSKeyedUnArchiver.unserializeNSKeyedArchiver(persistedkey)
Out[3]:
{'keyTag': 'T+PATGnEu5AstGkQ',
 'masterKey': b'\xd7~\x19\xf4]\xae\xbb\xa9\xf0\xdc2\xa0\x1b`\xf8\xab\xb9\xc9\xc6\x13P\xa3\xc6\xc4\xdf\x94k%M\x82\xbe,',
 'userId': '32d46086-2da6-4e9a-9872-5781ef4a2170',
 'passphrase': b'1\x8e\xd7\x86T\xd7J\xeb\x19}[\xe8\x96\xc5\xa6\x81p\xaaq\x8eC\xc0\xbaB\x98x\xb7\xd0\xb3\xc3G\xdc',
 'initializationVector': b'F\x88\xd8e\xfc\x90\xa1\x94\xf5\xccn\xf2\xc7Z\xbc\xbd'}
In [4]: NSKeyedUnArchiver.unserializeNSKeyedArchiver(persistedkey).get('masterKey').hex()
Out[4]: 'd77e19f45daebba9f0dc32a01b60f8abb9c9c61350a3c6c4df946b254d82be2c'
In [5]: NSKeyedUnArchiver.unserializeNSKeyedArchiver(persistedkey).get('initializationVector').hex()
Out[5]: '4688d865fc90a194f5cc6ef2c75abcbd'
```

The `masterkey` and `initializationVector` decrypts every memory key and iv..

(AES CBC)

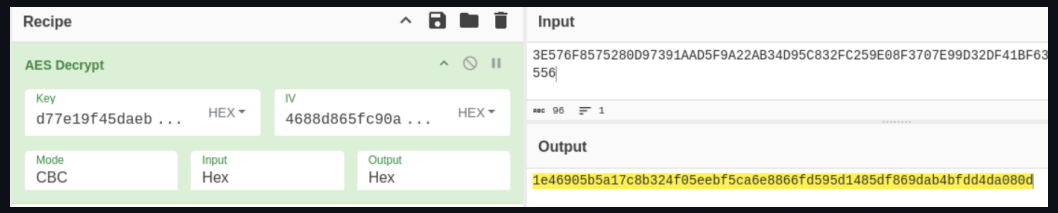
POLITIET
KRIPOS

Another layer of AES CBC..

Get the encrypted jkey and iv from gallery.encrypteddb

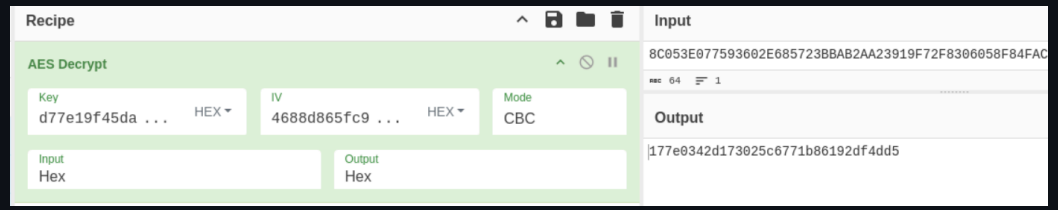
```
sqlite> SELECT hex(key),hex(iv) FROM snap_key_iv WHERE snap_id = '60A5C5A3-0EBB-4DC8-9DB9-A150F94BF4D5';
E1849B5A3E8E846BF69F9C7A9B2129DDC93D84310EB678ED00A4C52C4D28AD5E0425948FCD4D02AEB0E74A5554283264|24C18EEA70DC9934B54899C19A6FD19B2D36131520BB262782C83F060C5EAD30
```

Decrypt the uniqe MEO memory key with the KEK/persistedkey and IV



`37d0fb78b8bb0d4c48de7204def330e36643d225ca60398da51350675cace290`

Decrypt the uniqe MEO memory IV with the KEK/persistedkey and IV



`94d44b314cf7d5b3b624f00d7e2dff62`

POLITIET
KRIPOS

Decrypt https://cf-st.sc-cdn.net/h/fjAGkc2oIJEnAvVKL3t84?bo=Eg0aABoAMgEISAJQHGAB&uc=28

Recipe

AES Decrypt

Key: 3da51350675cace290 HEX

IV: b3b624f00d7e2dff62 HEX

Mode: CBC/NoPadding

Input: Raw

Output: Raw

Render Image

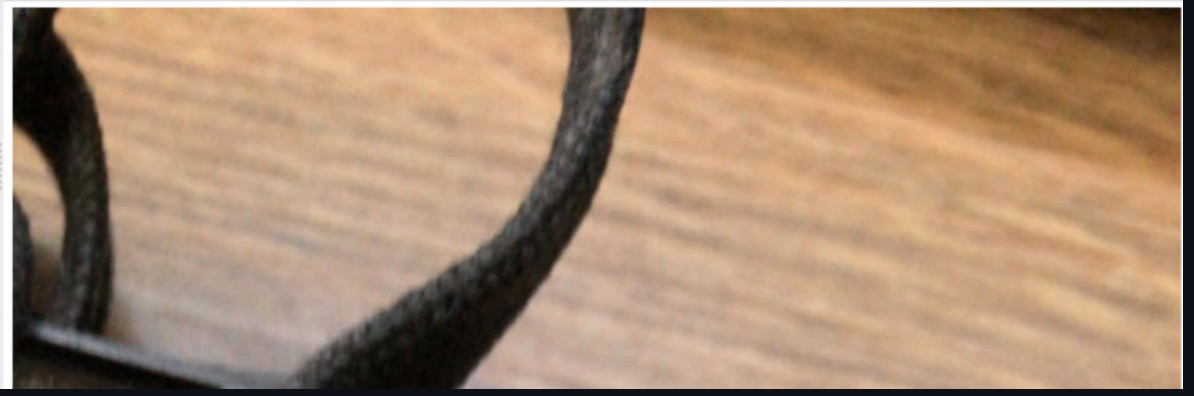
Input format: Raw

Input

total:
loaded:

122960 488

Output



POLITIET
KRIPOS

Decrypting memories on 🤖

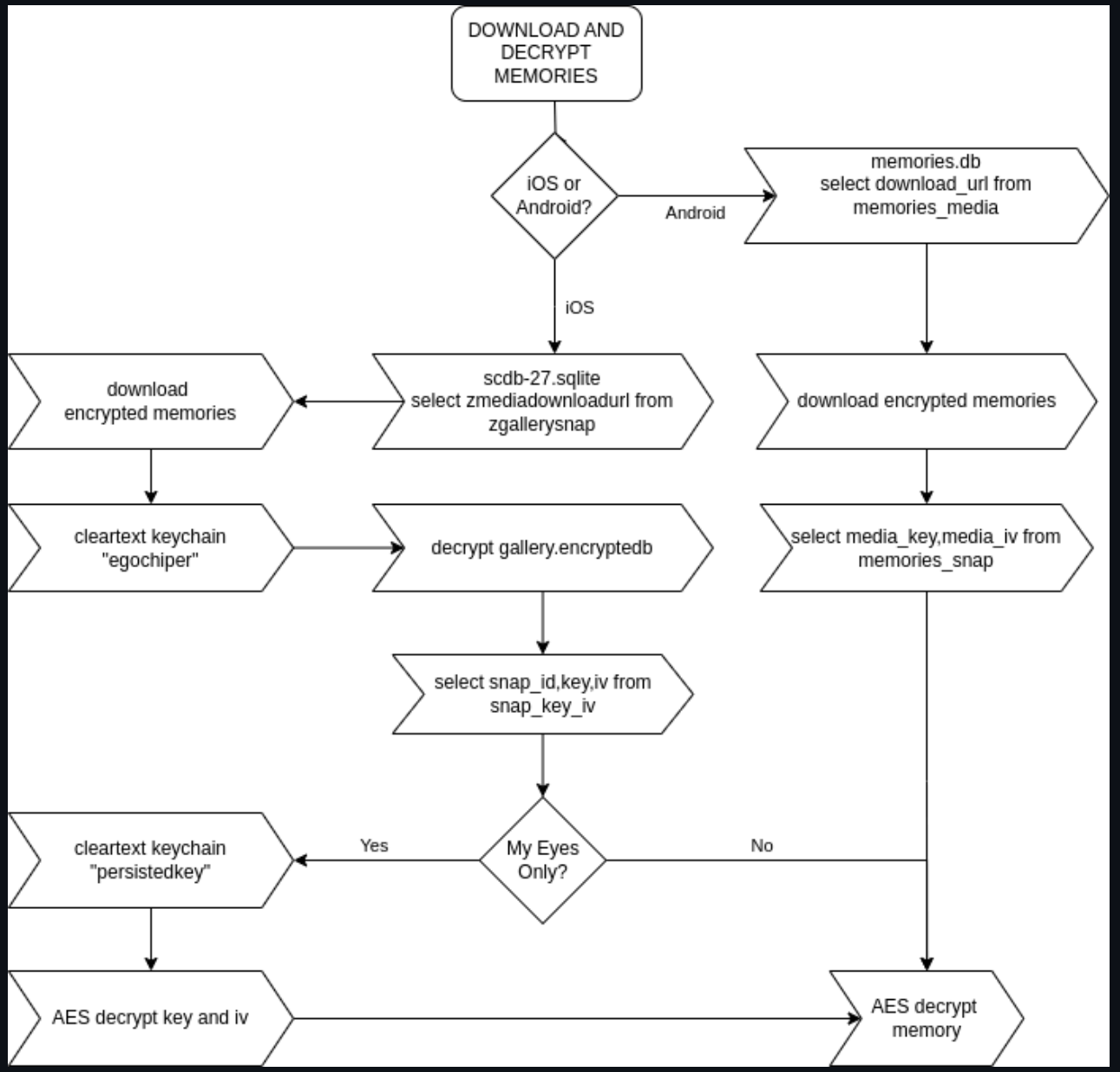
POLITIET
KRIPOS

On

It's almost the same but easier (no android keystore in use ❤️)

```
# Same key and iv as iOSs `com.snapchat.keyservice.persistedkey`
$ sqlite3 memories.db "SELECT hex(base64(master_key)),hex(base64(master_key_iv)) FROM memories_me
o_confidential;"
D77E19F45DAEBBA9F0DC32A01B60F8ABB9C9C61350A3C6C4DF946B254D82BE2C|
4688D865FC90A194F5CC6EF2C75ABCBD


# Same url as iOS `scdb-27.sqlite3`
$ sqlite3 memories.db "SELECT download_url  FROM memories_media WHERE
_id='60A5C5A3-0EBB-4DC8-9DB9-A150F94BF4D5';"
https://cf-st.sc-cdn.net/h/fjAGkc2oIJEnAvVKL3t84?bo=Eg0aABoAMgEISAJQHGAB&uc=28


$ curl -o 60A5C5A3-0EBB-4DC8-9DB9-A150F94BF4D5.android.bin https://cf-st.sc-cdn.net/h/fjAGkc2oIJEnAvVKL3t84?bo=Eg0aABoAMgEISAJQHGAB&uc=28

# Same key and IV as iOS, however the key and IV is not encrypted with the master_key or master_key_iv..... shhhh
$ sqlite3 memories.db "SELECT hex(base64(media_key)),hex(base64(media_iv)) FROM memories_snap WHERE
_id == '60A5C5A3-0EBB-4DC8-9DB9-A150F94BF4D5';"
37D0FB78B8BB0D4C48DE7204DEF330E36643D225CA60398DA51350675CACE290|
94D44B314CF7D5B3B624F00D7E2DFF62

$ md5sum 60A5C5A3-0EBB-4DC8-9DB9-A150F94BF4D5.android.bin
3aa3ff5a6294a75e7a688ff57bd0adda
$ md5sum 60A5C5A3-0EBB-4DC8-9DB9-A150F94BF4D5.ios.bin
3aa3ff5a6294a75e7a688ff57bd0adda
```

POLITIET
KRIPOS



POLITIET
KRIPOS

Undelete 🗑

POLITIET
KRIPOS

Recover deleted memories

```
$ sqlite3 scdb-27.sqlite3 'PRAGMA journal_mode;'
wal
```

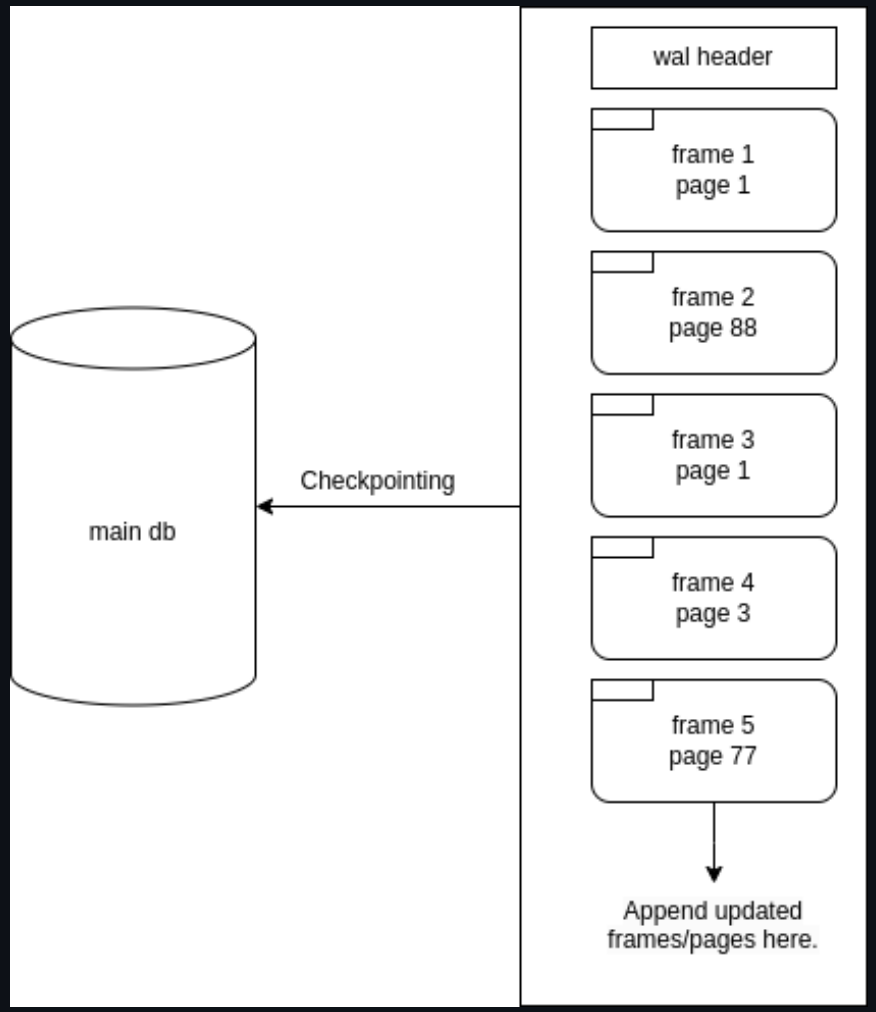
```
sqlcipher ../../gallery.encrypteddb
sqlite> PRAGMA key="x'242927b1d7424f5c786cc323f7034ba7c158a6fa9c71255445e16b054ebfd8a9'";
ok
sqlite> PRAGMA cipher_page_size=1024;
sqlite> PRAGMA kdf_iter=64000;
sqlite> PRAGMA cipher_hmac_algorithm=HMAC_SHA1;
sqlite> PRAGMA cipher_kdf_algorithm=PBKDF2_HMAC_SHA1;
sqlite> PRAGMA journal_mode;
wal
```

POLITIET
KRIPOS

SQLite Write-Ahead Log (WAL)

TL;DR: wal usage makes it possible to recover deleted URLs, keys and IVs from the databases.

- Modifications (INSERT, DELTE, UPDATE) are written to the WAL file and not directly to the database to speed things up and handle abrupt exits.
- Checkpointing usually happens when the WAL file exceeds 1000 pages or when a database connection closes. 1000 pages means a ~4 MB large WAL file for Snapchat!

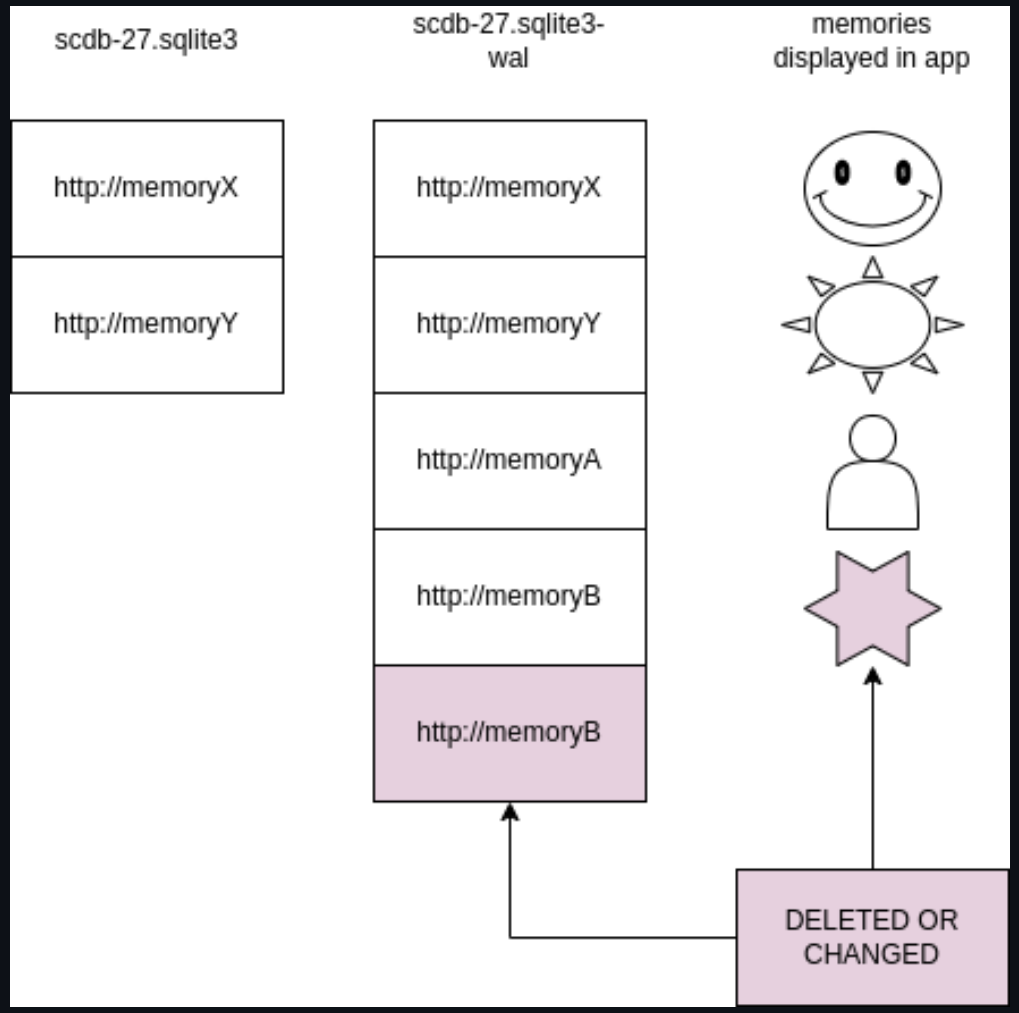


POLITIET
KRIPOS

Roll-forward journal

The WAL file acts as a roll-forward journal, storing all changes—including row deletions—that have been committed but not yet applied to the main database file (checkpointing).

```
PRAGMA wal_checkpoint; #forces a checkpoint
```



POLITIET
KRIPOS

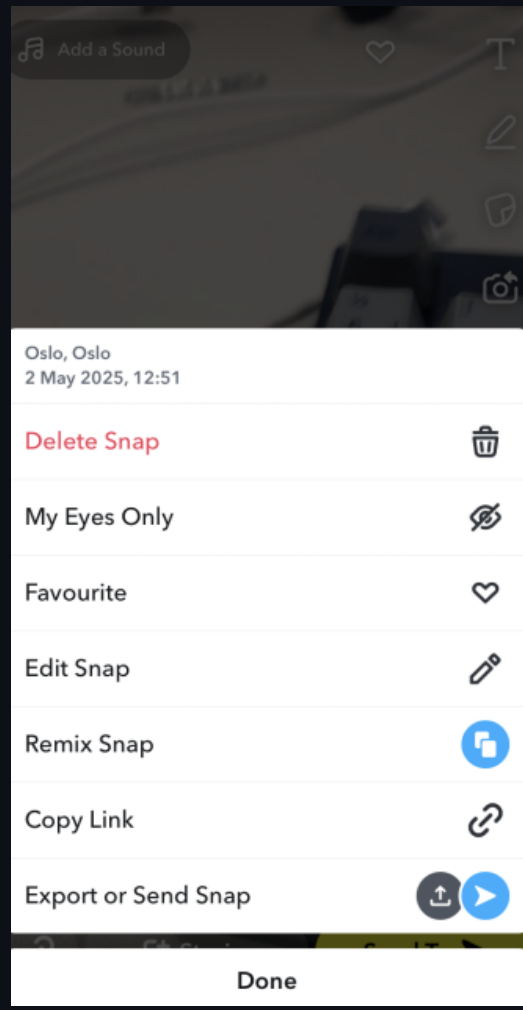
Example on

1. Use the app and delete a MEO memory.
2. Create database snapshot of every commit frame from the wal files of `gallery.encrypteddb` and `scdb-27.sqlite3`

```
./wal_versions.py --db gallery.encrypteddb --wal gallery.encrypteddb-wal  --outfolder wall_versions/
./wal_versions.py --db scdb-27.sqlite3 --wal scdb-27.sqlite3  --outfolder wall_versions/
```

3. In this experiment we get 8 different versions of `gallery.encrypteddb` and 41 versions of `scdb-27.sqlite3`. Then we decrypt each version of `gallery.encrypteddb`

```
decrypt_dbs.py --db gallery.encrypteddb* --key 242927b1d7424f5c786cc323f7034ba7c158a6fa9c71255445e16b054ebfd8a9
```



POLITIET
KRIPOS

Example on cont.

4. Diff the original db with the version with the last commit frame from the WAL:

```
$ sqldiff gallery.encrypteddb_c8.decrypted.db gallery.encrypteddb_c0.decrypted.db --table snap_key_iv --summary
snap_key_iv: 0 changes, 0 inserts, 2 deletes, 335 unchanged

$ sqldiff scdb-27.sqlite3_c45 scdb-27.sqlite3_c00 --table zgallerysnap --summary
zgallerysnap: 0 changes, 3 inserts, 19 deletes, 130 unchanged
```

```
$ sqldiff gallery.encrypteddb_c8.decrypted.db gallery.encrypteddb_c0.decrypted.db --table snap_key_iv
DELETE FROM snap_key_iv WHERE rowid=336;
DELETE FROM snap_key_iv WHERE rowid=337;
$ sqlite3 gallery.encrypteddb_c8.decrypted.db "SELECT snap_id,hex(key),hex(iv),encrypted FROM snap_key_iv WHERE rowid=336"
2E39E633-ACFA-480F-848B-DCDD177EC4E0|
C040AB99959A506490E6864E31E2628A71C0E5D1AEA1AE2A3FAD1070B21886C3|
320ED0DE7B37835A2F4F9C78B12A0FA0|
0
$ sqlite3 gallery.encrypteddb_c8.decrypted.db "SELECT snap_id,hex(key),hex(iv),encrypted FROM snap_key_iv WHERE rowid=337"
919CE1B6-423D-4C05-948E-609E1642505B|
F469B5BB1FB95A88DDBE0B5F168E90D9BF9460B89FCD5F71027AB23D29D2989D89DD35843E0C41FE169A884290C2D8D8|
83261D88C789DD2D7482E9E9D4E9A275D7082DB53F82D14D130683128FCA6E35
|1
```

Looks like one MEO memory and one regular memory has been deleted?!

POLITIET
KRIPOS

Example on cont.

5. Get the memories URL:

```
$ sqlite3 scdb-27.sqlite3_c45 "SELECT zmediadownloadurl FROM zgallerysnap WHERE zsnapid == '2E39E633-ACFA-480F-848B-DCDD177EC4E0'
OR zmediaid == '2E39E633-ACFA-480F-848B-DCDD177EC4E0';"

https://cf-st.sc-cdn.net/h/3eH3hDhTzfXqFHdTYHVjg?bo=EgkyAQhIAVALYAE%3D&uc=11

$ sqlite3 scdb-27.sqlite3_c45 "SELECT zmediadownloadurl FROM zgallerysnap WHERE zsnapid == '919CE1B6-423D-4C05-948E-609E1642505B' OR
zmediaid == '919CE1B6-423D-4C05-948E-609E1642505B';"

https://cf-st.sc-cdn.net/h/3eH3hDhTzfXqFHdTYHVjg?bo=EgkyAQhIAVALYAE%3D&uc=11
```

hmm identical URLs!?

This is how snapchat moves a normal memory to a MEO memory. Since I deleteded a MEO memory it's key and IV is now encrypted in the latest database commit...

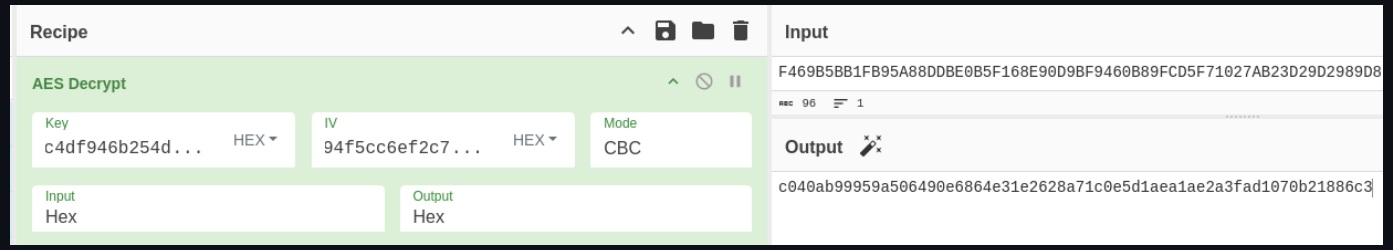
```
curl -o 919CE1B6-423D-4C05-948E-609E1642505B.bin https://cf-st.sc-cdn.net/h/3eH3hDhTzfXqFHdTYHVjg?bo=EgkyAQhIAVALYAE%3D&uc=11
$ file 919CE1B6-423D-4C05-948E-609E1642505B.bin
919CE1B6-423D-4C05-948E-609E1642505B.bin: data
```

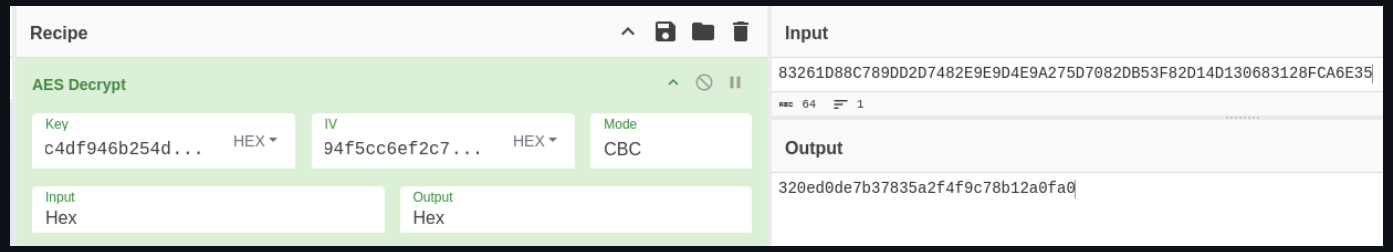
POLITIET
KRIPOS

Example on cont.

6. No need to decrypt the key and IV since we have it's 'cleartext' in the recovered data from the WAL:

```
$ sqlite3 gallery.encrypteddb_c8.decrypted.db "SELECT snap_id,hex(key),hex(iv),encrypted FROM snap_key_iv WHERE rowid=336"
2E39E633-ACFA-480F-848B-DCDD177EC4E0|
C040AB99959A506490E6864E31E2628A71C0E5D1AEA1AE2A3FAD1070B21886C3|
320ED0DE7B37835A2F4F9C78B12A0FA0|
0
$ sqlite3 gallery.encrypteddb_c8.decrypted.db "SELECT snap_id,hex(key),hex(iv),encrypted FROM snap_key_iv WHERE rowid=337"
919CE1B6-423D-4C05-948E-609E1642505B|
F469B5BB1FB95A88DDBE0B5F168E90D9BF9460B89FCD5F71027AB23D29D2989D89DD35843E0C41FE169A884290C2D8D8|83261D88C789DD2D7482E9E9D4E9A275D7082DB53F82D14D130683128FCA6E35
|1
```



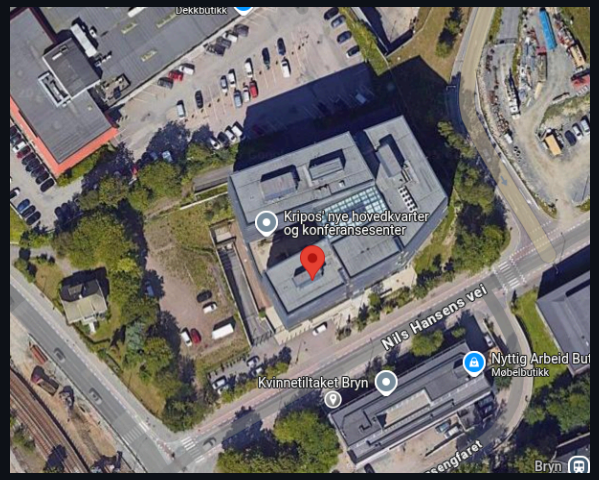


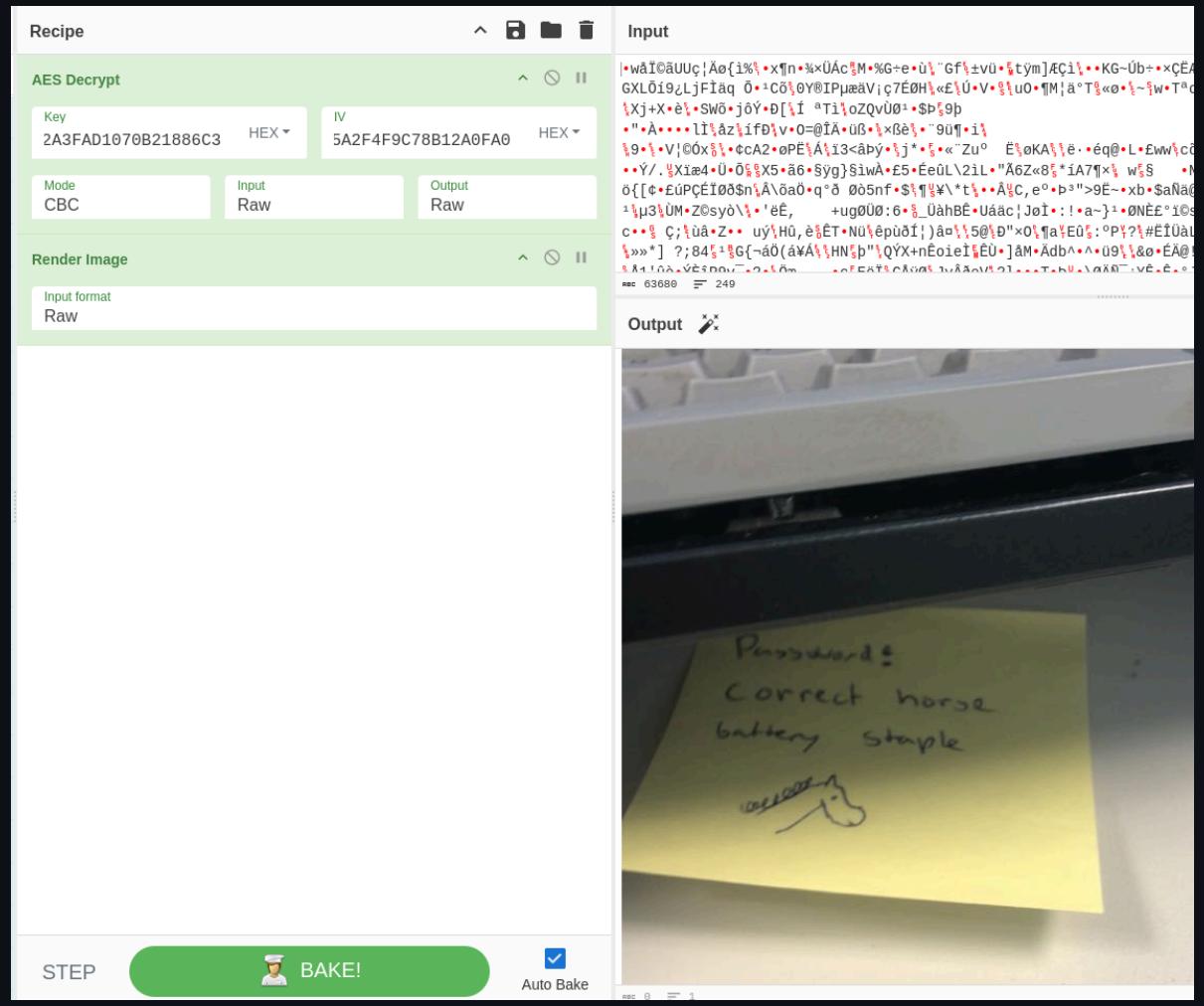
POLITIET
KRIPOS

Example on cont.

7. Finally recover the memory

```
sqlite3 gallery.encrypteddb_c8.decrypted.db
"SELECT latitude, longitude FROM snap_location_table
WHERE snap_id == '919CE1B6-423D-4C05-948E-609E1642505B';"
59.9088858237803|10.8171173813676
```





POLITIET
KRIPOS

Results

My Eyes Only

- The MEO pin/password are possible to crack:
  - On we need the keychain and two databases.
  - On we need a database.
- The MEO pin/password are not used to protect the confidentiality of the MEO memories (it's only a GUI thing..).
  - On a static AES key and IV in the keychain is used to encrypt MEO keys and IVs in `gallery.encrypteddb`. URLs are found in `scdb-27.sqlite`
  - On AES keys, IVs and URLs are found in the `memories.db` file

Memory recovery results

- Deleting a memory only deletes the URL for it and its key and IV pair from the databases
  - Memories are keep on the servers for some time after deletion (observed ~2 months)
  - No authenication is necessary to download encrypted Memories (curl works).

Memory recovery results

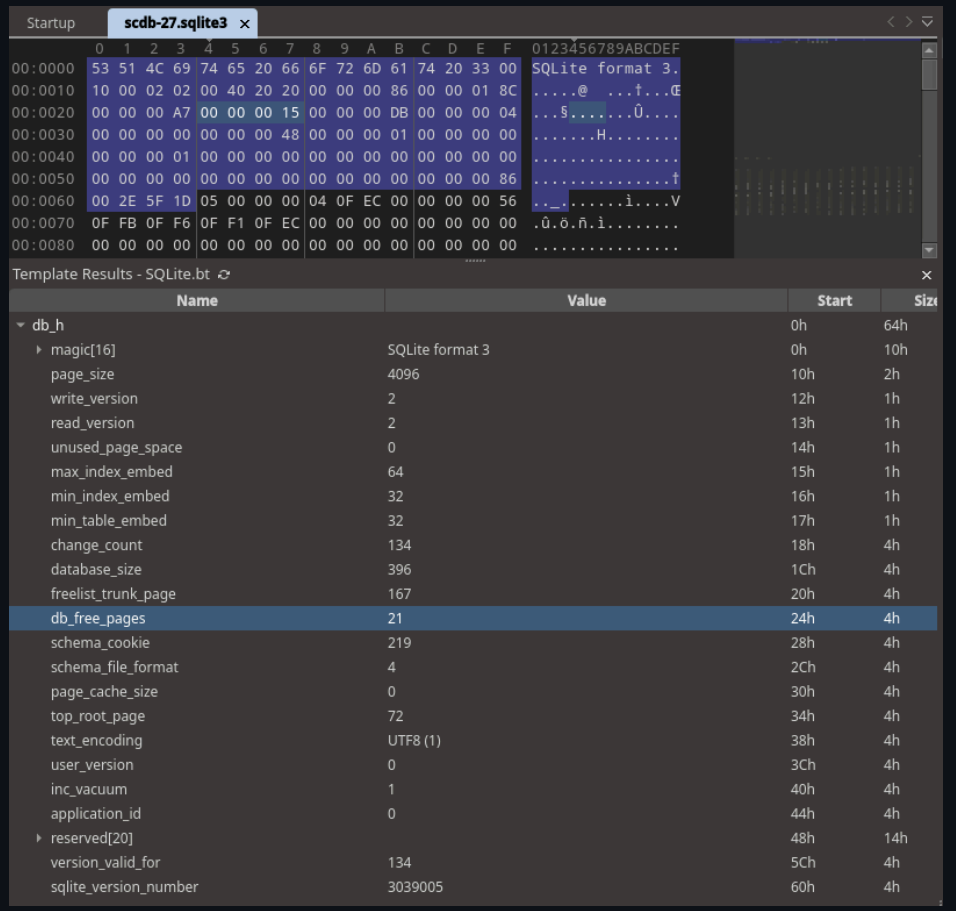
- The use of SQLite journaling Write-Ahead Log (WAL) makes it possible to recover deleted URLs and key material in the databases.

POLITIET
KRIPOS

Bonus: More possibilities for database recovery

- SQLite free pages: Deleted pages (4096 bytes each) in the database are not immediately removed from the database file. Instead, they are added to a free list. In every page in the free list the old deleted data remaining intact until the pages are reused (e.g when the database grows).

```
# tool from USA DoD Cyber Crime Center (DC3)
$ pip install sqlite-dissect
$ sqlite_dissect --carve --carve-freelists -k scdb-27.sqlite3 -d results
```



POLITIET
KRIPOS

Further research

- Scripting..
- How does Snapchat store the MEO pin/password online?
- Other artifacts like cache etc.?

POLITIET
KRIPOS

Questions?

`ole.martin.dahl@politiet.no`

Writeup available here: https://www.politiet.no/kripos-espor