A Fast Recovery of Encrypted Message Database of WeChat On GPU

Fei Yu*, Yu Shi, Hao Yin
Science and Technology on Communication Security Laboratory
Chengdu, China
e-mail: feiyu80@gmail.com, sherryxupt1118@163.com, 4181739@qq.com

2020 IEEE 5th International Conference on Signal and Image Processing (ICSIP) | 978-1-7281-6896-8/20/$31.00 ©2020 IEEE | DOI: 10.1109/ICSIP49896.2020.9339398

***Abstract*—WeChat is one of the most popular instant-messaging applications in the world. The recovery of WeChat's encrypted local database is concerned by forensic community. This paper studies a fast recovery method in a special situation that we only have the encrypted database without any helpful information. The new method uses a fast implementation of pbkdf2 function which is two times faster than before. The new method could successfully recover the database in about 30 seconds on our GPU system, while the previous method will take about 10 days on a desktop CPU.**

Keywords-WeChat; database decryption; digital forensic; GPU

I. INTRODUCTION

WeChat is one of the most popular instant-messaging (IM) applications in the world. WeChat can run in most platforms such as android, iOS, Windows, MacOS and Linux (by WEB). People use WeChat to communicate with others by sending text, image, voice and video messages. Besides, online payment, "Moments", public account and many public social services make WeChat be the main social network application. According to the annual report of Tencent company, there are 1,164.8 Million active users of WeChat by the end of 2019, with a 6.1% rise compared with 2018[1].

On the other hand, more and more criminal activities used this tool. Therefore, it has attracted the attention of the forensic community [2]-[6]. To recovery and analysis the message records is the key work. The message records is encrypted and stored in a SQLite database. The encrypted key is derived from the device's IMEI/MEID and the user's ID named UIN. Researchers have designed some recovery methods [4], [5] which will be reviewed in section II.B. However, in a special situation that we can't obtain the IMEI/MEID and UIN, the recovery performance is very low which will take about 10 days.

This paper will study the recovery of WeChat's local message database in the above special situation. In section I we will review the local message database and some recovery methods. In section II we will analysis the detail of the recovery (decryption) progress and the performance of the PBKDF2 function, and propose an optimized recovery method which will be two times faster than before. In section III we will implement the new method on GPU. Our experiment shows that the optimized method will recover an encrypted WeChat's local message database in about 30 seconds on our GPU system without any more information other than the database itself.

II. RECOVERY OF WECHAT'S LOCAL MESSAGE DATABASE

A. *Wechat's Local Message Database*

The chat records of WeChat are stored in a SQLite database file named "EnMicroMsg.db" which is encrypted and could be found in the corresponding personal data folder.

Take android system for example, all the application data will be stored in two paths: "/data/data/com.tencent.mm/" and "/sdcard/Tencent/MicroMsg". The local databases, application configurations, and other important data are stored in three subfolders of "/data/data/com.tencent.mm/": "databases", "shared_prefs" and "MicroMsg". For each user, its personal data folder will be placed under the "MicroMsg" subfolder, whose name is computed from a unique 32-bit number "UIN" by MD5 algorithm:

$$\text{<udir>} = \text{MD5}(\text{'mm'} \parallel \text{UIN}).$$

where the symbol <udir> denotes the personal data subfolder name, 'mm' is the literal string, "||" means string concatenation. For example, suppose the UIN is 0x12345678, then the personal data folder should be: "/data/data/com. tencent.mm/MicroMsg/d27a704d3286180f19a358557358be a9", where the user's local message database "EnMicroMsg.db" could be found.

The local message database "EnMicroMsg.db" is a SQLite type database. SQLite is the most widely deployed database in the world for its advantage such as self-contained, serverless and zero-configuration. The main database file consists of at least one page with fixed size between 512 bytes and 32678 bytes. The page size of WeChat is 1204 bytes. The first 100 bytes of the database file is the file header, which is described in the Table 1. Note that all the multi-byte data are stored with the big-endian mode.

WeChat's local message database is entirely encrypted by an open source encryption tool "sqlcipher" including the file header. The encryption algorithm is AES-256 with CBC mode where the IV could be extracted from the encrypted file. The encryption key is derived from a 7 bytes password and a 16-bytes randomly generated salt by using PBKDF2-HMAC-SHA1 function with 4000 iterations. The password is computed easily:

$$\text{password} = \text{Left7}(\text{MD5}(\text{IMEI/MEID} \parallel \text{UIN})),$$

where the Left7 function extracts the first seven bytes of a string.

TABLE I. THE SQLITE FILE HEADER [7]

Offset (byte)	Size (bytes)	Description
0	16	Fixed header string: "SQLite format 3\000"
16	2	The database page size in bytes, between 512 and 32768.
18	1	File format write version. 1 for legacy; 2 for WAL.
19	1	File format read version. 1 for legacy; 2 for WAL.
20	1	Bytes of unused at the end of each page. Default 0.
21	1	Maximum embedded payload fraction. Must be 64.
22	1	Minimum embedded payload fraction. Must be 32.
23	1	Leaf payload fraction. Must be 32.
24	4	File change counter.
28	4	Size of the database file in pages.
32	4	Page number of the first freelist trunk page.
36	4	Total number of freelist pages.
40	4	The schema cookie.
44	4	The schema format number.
48	4	Default page cache size.
52	4	The page number of the largest root b-tree page.
56	4	The database text encoding. 1for UTF-8, 2 for UTF-16le, 3 for UTF-16be.
60	4	The "user version".
64	4	Non-zero for incremental-vacuum mode. Zero otherwise.
68	4	The "Application ID".
72	20	Reserved. Must be zero.
92	4	The version-valid-for number.
96	4	SQLITE_VERSION_NUMBER

B. Recovery of the Database

To recovery the WeChat's local database, we need IMEI/MEID and UIN or just the 7-bytes password to compute the decryption key. In practice, we face three situations: we know the IMEI and UIN [4], we only know the IMEI [5], or we know nothing but the encrypted database [4].

In the first case, the mobile device has been rooted, we need to decrypt the last logged user's local database. We can obtain the IMEI by tapping *#06# in the dial interface, and extract the user id UIN from the file "/data/data/com. tencent.mm/MicroMsg/systemInfo.cfg". Note that there may exist more than one IMEI for some devices such as iPhone XR. For these devices we should use MEID instead of IMEI. Once we obtain the IMEI/MEID and UIN, we can compute the decryption key and decrypt the local database of WeChat as shown in Fig.1.

In the second case, we can't directly extract the UIN from any configuration file. WeChat only saved the last logged user's UIN. If we want to decrypt any other user's local database, we have to identify the user's personal folder and then compute the UIN from its name, i.e. the <udir>. There are two methods to compute the UIN based on the <udir>. The first method is exhaustively searching the UIN in 32 bit space, which will take about more than 48 hours on a desktop PC. The second method is precomputing a table which stores all the 2^{32} UINs and the names. The table requires about 100 GB of storage, and the table query will take some seconds. The two key derived methods are shown in Fig. 2a) and Fig.2 b), respectively.

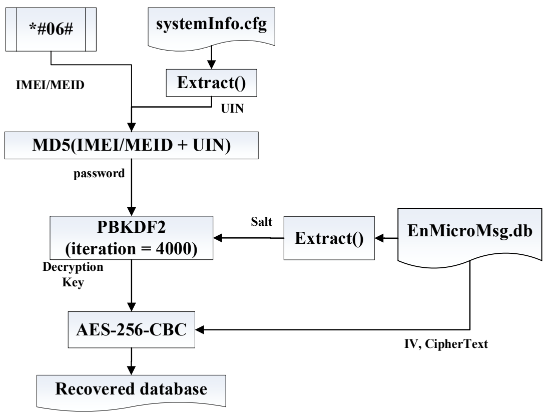

Figure 1. The decryption process in case 1

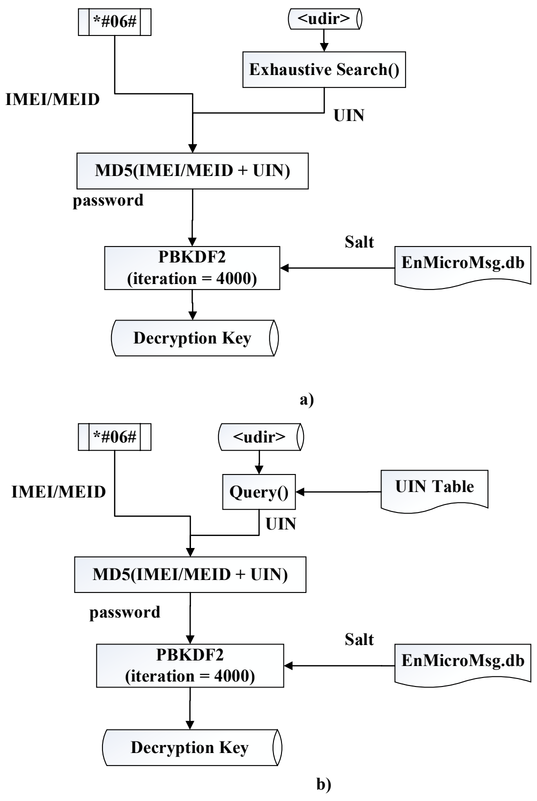

Figure 2. The decryption key derived process in case 2

The third case is more complicated, we only have the encrypted local database. We have no more helpful information such as the IMEI/MEID, UIN and the <uin>. Let's review the entirely encryption process. WeChat uses the 7-bytes password and a 16-bytes salt to derive the encryption key, it will divide the local database in blocks (pages) of 1024 bytes, and then it encrypts the blocks by AES-256-CBC algorithm individually. The salt of PBKDF2 and the IV of CBC mode were randomly generated and they were stored in the head and the tail of the first page respectively. Now in the process of decryption, WeChat will extract the salt from the first 16 bytes of the page, generate the encryption key, extract the IV from the last 16 bytes of the page, decrypt the other data, and pad the fixed header string in the head. A study shows that the 17 to 24 byte of the plaintext of the first page are always " 0x40, 0x00, 0x01, 0x01, 0x00, 0x40, 0x20, 0x20"[4]. According Table I and our experiments, the 8 bytes may be "0x40, 0x00, 0x02, 0x02, 0x10, 0x40, 0x20, 0x20" in the new version of WeChat. Now, the recovery process turns into a password cracking process. The above 8 bytes data become the checksum of the cracking process. The password cracking process is shown in Fig. 3 which will take about 10 days on a desktop PC.

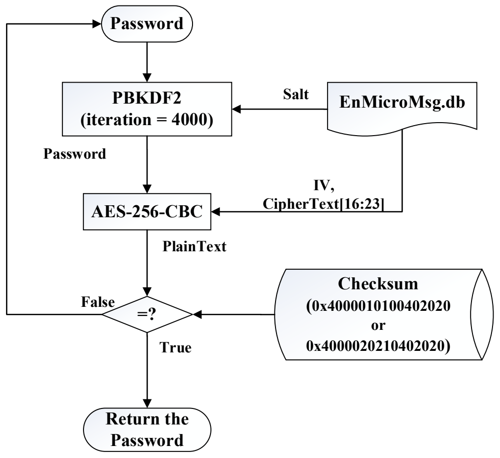

Figure 3. The password cracking process in case 3

III. A FAST RECOVERY

In this section we will introduce a fast recovery method of WeChat's local database in the third case of subsection II.B.

A. *PBKDF2 Function*

In the above password cracking process of subsection II.B, PBKDF2 (Password Based Key Derived Function 2) is the most complicated step for its 4000 iterations. PBKDF2 is a part of PKCS#5 (Public Key Cryptography Standard #5), which is provided by RSA company [8]. PBKDF2 is widely used in the world such as WPA/WPA2 [9], 1password [10], and WeChat.

The following is the pseudocode of PBKDF2.

```
ALGORITHM 1 - PBKDF2
INPUT: pwd, salt, iteration, klen,hlen
OUTPUT: Key
1.  r = CEIL( klen/hlen)
2.  FOR i FROM 1 TO r DO
      a) T = HMAC(pwd, salt || i)
      b) U = T
      c) FOR j FROM 1 TO iteration DO
           i.  T = HMAC(pwd, T)
           ii. U = U ⊕ T
      d) out = out || U
3.  Key = LEFT(klen, out)
```

klen is the key length we need, *hlen* is the length of the output data of HMAC, CEIL() is the ceiling function, ⊕is the ANDOR operation, LEFT(*l*, d) is the truncation function which truncates the left *l* bytes of d.

HMAC function HMAC (*key*, *text*) will pad the input *key* to *K* by zero bytes if the length of *key* is not greater than the block length of the hash function, otherwise hash the key. HMAC computes H(K⊕ *opad*, H(K⊕ *ipad*, text)) as its output, where *opad* and *ipad* are fixed constants, H is the hash function.

For WeChat， the above parameters are: H = SHA1, klen = 32, hlen = 20, iteration = 4000.

B. *The Fast Recovery Algorithm*

Now we give a detailed analysis of the PBKDF2 in WeChat. Note that the length of the password is just seven bytes which is less than the block length of SHA1. So the step 2.c) of PBKDF2 can be rewritten as the following c').

```
c') FOR j FROM 1 TO iteration DO
      i.   K = PADDING (pwd)
      ii.  T = SHA1( K⊕opad, SHA1(K⊕ipad, T))
      iii. U = U ⊕ T
```

Note that *T* is the output of SHA1, its length is 20 bytes which is less than 64 bytes, therefore step c') can be rewritten as the following c'') and the length of every input of SHA1 here is 64 bytes. That means they will compute only one block. The total computation amount is about *O(4*iteration)* SHA1 blocks.

```
c'') FOR j FROM 1 TO iteration DO
       i.   K = PADDING (pwd)
       ii.  X = SHA1(K⊕ ipad)
       iii. Y = SHA1(X || T)
       iv.  Z = SHA1(K⊕ opad)
       v.   T = SHA1(Z || Y)
       vi.  U = U ⊕ T
```

Furthermore, the value of K is the same in each iteration. So we can move the above c'')i, c'')ii and c'')iv out of the iteration. And we obtained the optimized PBKDF2 as the following pseudocode.

```
ALGORITHM 2 – Optimized PBKDF2 of WeChat
INPUT: pwd, salt
OUTPUT: Key
1.  r = 2
2.  K = PADDING (pwd)
3.  Precompute:
      a) X = SHA1(K⊕ ipad)
      b) Z = SHA1(K⊕ opad)
4.  FOR i FROM 1 TO 2 DO
      a) T = salt || i
      b) Y = SHA1(X || T)
      c) T = SHA1(Z || Y)
```

```
    d) U = T
    e) FOR j FROM 1 TO 4000 DO
        i.   Y = SHA1(X || T)
        ii.  T = SHA1(Z || Y)
        iii. U = U ⊕ T
    f) out = out || U
5.  Key = LEFT(32, out)
```

It's clear that the amount of hash computation in the major iteration step reduced from 4 to 2 in algorithm 2. Hence the total computational complexity reduced from about *O(4*iteration)* SHA1 blocks to about O(2*iteration) SHA1 blocks. The performance of the password cracking algorithm will be doubled at the same time. We also indicate that the fast algorithm will not affect the security of PBKDF2. The key point is the iteration number. If we use a large iteration number in PBKDF2 like the famous password management software 1password which adopts more than 100000 iterations, the security is still guaranteed. A recent security analysis of PBKDF2 could be seen in [11].

When we replace algorithm 1 in Fig.3 with algorithm 2, we will get the fast recovery algorithm of WeChat's Encrypted Message Database. Compared with the previous algorithm, the new algorithm has twice the performance.

C. *Implementation on GPU*

Now we will implement the optimized recovery algorithm on GPU systems.

In recent years, GPU (Graphic Processor Unit) has developed rapidly in general computing because of its parallel computing characteristics and programmability, high performance, low power consumption and other advantages. GPU has hundreds of computing units (named cuda cores for NVIDIA's GPU). GPU is naturally suitable for computing intensive operations such as hash computing and password cracking [12]. A brief comparison between CPU and GPU architecture is shown in Fig.4.

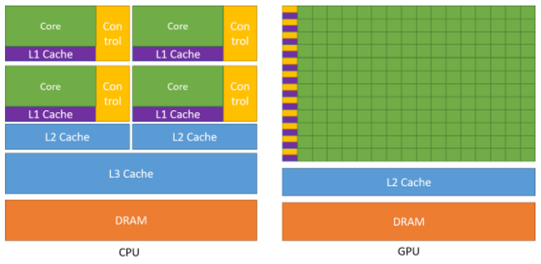

Figure 4. Comparison between CPU and GPU

There are two parallel computing platform and programming model for general computing on GPU: OpenCL which is developed by Khronos Group for all GPUs and CUDA which is developed by NVIDIA for NVIDIA's GPUs. However, CUDA is favored by developers because of its friendly and convenient programming [13], [14]. Therefore we implement the optimized recovery algorithm on NVIDIA's GPU and CUDA platform.

We developed a three-layer parallel GPU system for password cracking which is composed by multiple computing nodes. Each node has multiple GPU cards. The three layers are the task management layer which manages the whole system, the GPU management layer which manages the GPUs on a node and the GPU computing layer which implements the password cracking task. The architecture of our system is shown in Fig. 5.

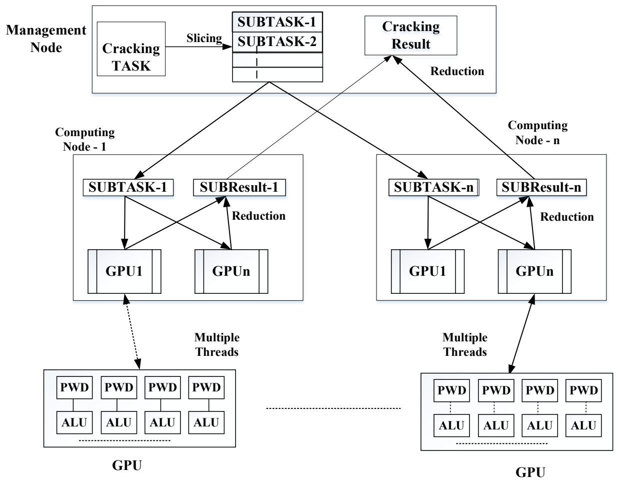

Figure 5. The 3-Layer GPU system architecture

Some more optimization methods should be employed on GPU. First, the length of the input password candidates the salts is unique. So we can limit the memory space of one thread to get more threads. In our implementation, the execution configuration was set as <<256, 2176>>. That means, there are 557,056 threads can be executed in parallel on one GPU card.

Secondly, the charset of the password are fixed, that is {0-9A-F}. Thus the password space are fixed whose size is 16^7= 268435456. So we can optimize the task slicing and assignment to avoid the communication latency.

Finally, the most important optimization is the memory accessing optimization. There are two kinds of memory on GPU: local memory and share memory. The former is large (GBs) but access is slow. The latter is very small (KBs) but access is very fast. So we have to allocate the memory space by sophisticated tricks to get the balance between space and speed. Other optimization techniques include CUDA memory consistency which could ensure memory accessing in parallel for multiple threads in one thread block and using of a special data type vector which could improve data throughput.

Our experimental system has two computing nodes. Each node has eight Geforce RTX 2080Ti cards which are the last generation of NVIDIA's flagship products. The specifications of RTX 2080Ti is shown in Table II. On our system, the fast recovery algorithm has very high performance: about 580,000 passwords/second per card. So one card could successfully recover the password in about 8 minutes and our system could do that in about 30 seconds. A comparison between the fast method and the original one is shown in Table III.

TABLE II. THE SPECIFICATIONS OF RTX 2080TI

GPU Architecture	Turing
NVIDIA CUDA Cores	4352
Base Clock	1350MHz
Memory Size	11GB
Memory Speed	14Gbps
Memory Bandwidth	616GB/s

TABLE III. PERFORMANCE COMPARISON BETWEEN THE FAST AND THE ORIGINAL METHODS

Platform	CPU(i7-2600, openmp)		GPU(1 * 2080Ti)	
	Speed (p/s)	MAX Recovery Time	Speed (p/s)	MAX Recovery Time
Orininal method	320	10 days	220,000	1221 seconds
Fast method	650	5 days	580,000	463 seconds

IV. CONCLUSION

In this paper we presented a fast recovery method of WeChat's encrypted local database without any information but the encrypted file itself. We reviewed the database structure, encryption and decryption process. We updated the cracking checksum in the new version of WeChat. Based on an optimized algorithm of PBKDF2, we introduced an optimized recovery method which is two times faster than before. Our experiments shows that the encrypted database could be successfully recovered in about 8 minutes on one GTX2080Ti GPU card or 30 seconds on our GPU system.

Our work could bring new tools and convenience to forensic work. On the other hand, it remind us that WeChat's local database is not security nowadays. The encryption should be strengthened such as increasing the iteration number and the password length.

ACKNOWLEDGMENT

This work was supported by the Applied Basic Research Project of Sichuan Province (NO. 2018JY0102).

The authors would like to thank the reviewers for their useful opinions to improve this paper

REFERENCES

[1] Tencent company, "The annual report of Tencent". 2020, [online] Available: https://www.tencent.com/zh-cn/investors.html

[2] D. Kao, T. Wang and F. Tsai, "Forensic Artifacts of Network Traffic on WeChat Calls," 2020 22nd International Conference on Advanced Communication Technology (ICACT), Phoenix Park, PyeongChang,, Korea (South), 2020, pp. 262-267, doi: 10.23919/ICACT48636.2020.9061437.

[3] F. Zhou, Y. Yang, Z. Ding and G. Sun, "Dump and analysis of Android volatile memory on Wechat," 2015 IEEE International Conference on Communications (ICC), London, 2015, pp. 7151-7156, doi: 10.1109/ICC.2015.7249467.

[4] L. Zhang, F. Yu and Q. Ji, "The Forensic Analysis of WeChat Message," 2016 Sixth International Conference on Instrumentation & Measurement, Computer, Communication and Control (IMCCC), Harbin, 2016, pp. 500-503, doi: 10.1109/IMCCC.2016.24.

[5] Songyang Wu, Yong Zhang, Xupeng Wang, Xiong Xiong and Lin Du, "Forensic analysis of WeChat on Android smartphones," Digital Investigation, vol. 21, June 2017, pp. 3-10, doi: 10.1016/j.diin.2016.11.002

[6] ZHANG Yanjiao, ZENG Guangyu, FENG Peijun and WANG Haoyuan, "Forensics and Analysis Method of WeChat Based On Android Smart Phone," Journal of Information Engineering University, vol. 19, no.6, Dec. 2018, pp. 719-725, doi: 10.3969/j.issn.1671-0673.2018. 06.015

[7] SQLite official website. "Database File Format", [online] Available: https://www.sqlite.org/fileformat.html

[8] "PKCS #5: Password-based cryptography specification version 2.0", no. 2898, 2000.

[9] "IEEE Std 802.11 i-2004", "Part 11: wireless LAN medium access control (MAC) and physical layer (PHY) specifications", 2004.

[10] "How PBKDF2 strengthens your master password", Jan. 2017, [online] Available: https://support.1password.com/pbkdf2/.

[11] A. Visconti, O. Mosnáček, M. Brož and V. Matyáš, "Examining PBKDF2 security margin—Case study of LUKS," Journal of Information Security and Applications, vol. 46, 2019, pp. 296-306, doi: 10.1016/j.jisa.2019.03.016.

[12] Lan Tian, Ji Qingbing, Yu Fei and Zhang Lijun, "Research and Implementation of MD5 Brute-force Cracking Algorithm based on CUDA," Communications Technology, vol. 46, no.12, Dec. 2013, pp. 62-65, doi: 10.3969/j.issn.1002-0802.2013.12.015

[13] NVIDIA company, "CUDA C++ Programming Guide", 2020, [online] Available: https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html

[14] Shane Cook, CUDA Programming, Morgan Kaufmann, 2012