Magdeburger Journal zur Sicherheitsforschung

Gegründet 2011 | ISSN: 2192-4260
Herausgegeben von Stefan Schumacher
Erschienen im Magdeburger Institut für Sicherheitsforschung
http://www.sicherheitsforschung-magdeburg.de/publikationen/journal.html

This article appears in the special edition »In Depth Security – Proceedings of the DeepSec Conferences«.
Edited by Stefan Schumacher and René Pfeiffer

BitCracker

The Bitlocker Password Cracker

Elena Agostini and Massimo Bernaschi

BitLocker is a full-disk encryption feature available in recent Windows versions. It is designed to protect data by providing encryption for entire volumes and it makes use of a number of different authentication methods. In this work we present a solution, named BitCracker, to attempt the decryption, by means of a dictionary attack, of memory units encrypted by BitLocker with a user supplied password. To that purpose, we resort to GPU (Graphics Processing Units) that are, by now, widely used as general-purpose coprocessors in high performance computing applications. BitLocker decryption process requires the execution of a very large number of SHA-256 hashes and also AES, so we propose a very fast solution, highly tuned for Nvidia GPU, for both of them. In addition we take the advantage of a weakness in the BitLocker decryption algorithm to speed up the execution of our attack. We benchmark our solution using the three most recent Nvidia GPU architectures (Kepler, Maxwell and Pascal), carrying out a comparison with the Hashcat password cracker. Finally, our OpenCL implementation of BitCracker has been recently released within John The Ripper, Bleeding-Jumbo version.

Keywords: BitLocker, Hash, SHA-256, AES, GPU, CUDA, Cryptographic Attack, Password Cracking

1 Introduction

BitLocker is a data protection feature that integrates with the Windows operating system and addresses the threats of data theft or exposure from lost, stolen, or inappropriately decommissioned computers. It offers a number of different authentication methods, like Trusted Platform Module, Smart Key, Recovery Password, user supplied password. BitLocker features a pretty complex proprietary architecture but it also leverages some well-known algorithms, like SHA-256 and AES. It is possible, and relatively easy (to this purpose, commercial tools are available (»Elcomsoft Forensic Disk Decryptor«, 2018)) to instantly decrypt disks and volumes protected with BitLocker by using the decryption key extracted from the main memory (RAM). In addition, it is also possible to decrypt for offline analysis or instantly mount BitLocker volumes by utilizing the escrow key (BitLocker Recovery Key) extracted from a user's Microsoft Account or retrieved from Active Directory.

If the decryption key can not be retrieved, the only alternative remains to unlock password-protected disks by attacking the password. The same commercial tools above mentioned, offer this as an option but in a quite generic form (*i.e.,*) without taking into account the specific features of BitLocker. Moreover, according to some comments[1], they may be also not fully reliable. The goal of the present paper is to describe our approach to attack BitLocker password-protected storage units. We carefully studied available information about BitLocker architecture and directly inspected several types of units in order to find out how to minimize the amount of work required to check a candidate password. The platforms we use for the attack are based on Nvidia GPUs and we carefully optimized the most computing intensive parts of the procedure achieving a performance that is, at least, comparable with that provided by well-known password crackers like Hashcat (»Hashcat«, 2018) for the evaluation of the SHA-256 digest function. However, the main goal of our work is not providing an alternative to Hashcat as a general framework for dictionary attacks but to offer the first open-source high performance tool to test the security of storage units protected by BitLocker using the user password and recovery password authentication methods.

2 BitLocker

BitLocker (formerly BitLocker Drive Encryption) is a full-disk encryption feature included in the Ultimate and Enterprise editions of Windows Vista and Windows 7, the Pro and Enterprise editions of Windows 8 and Windows 8.1, Windows Server 2008 and Windows 10. It is designed to protect data by providing encryption for entire volumes.

BitLocker can encrypt several types of memory units like internal hard disks or external memory devices [2](flash memories, external hard disks, etc..) offering a number of different authentication methods, like Trusted Platform Module, Smart Key, Recovery Key, password, etc.. In this paper we focus on two different authentication modes: the *user password mode*, in which the user, to encrypt or decrypt a memory device, must type a password (as represented in Figure 1) and the *recovery password mode*, that is a 48-digit key generated by BitLocker (regardless of the authentication method chosen by the user) when encrypting a memory device[3] . By means of the recovery password the user can access an encrypted device in the event that she/he can't unlock the device normally.

During the encryption procedure, each sector in the volume is encrypted individually, with a part of the encryption key being derived from the sector number itself. This means that two sectors containing identical unencrypted data will result in different encrypted bytes being written to the disk, making it much harder to attempt to discover keys by creating and encrypting known data. BitLocker uses a complex hierarchy of keys to encrypt devices. The sectors themselves are encrypted by using a key called the *Full-Volume Encryption Key* (FVEK). The FVEK is not used by or accessible to users and it is, in turn, encrypted with a key called the *Volume Master Key* (VMK). Finally, the VMK is also encrypted and stored in the volume; for instance, if the memory device has been encrypted with the user password method, in the volume metadata there are two encrypted VMKs: the VMK_U, that is the VMK encrypted with the user password, and the VMK_R, that is the VMK encrypted with the recovery password.

During the decryption procedure (Figure 2) BitLocker, depending on the authentication method in use, starts to decrypt the VMK. Then, if it obtains the right value for the VMK, it decrypts in turn the FVEK and then the entire memory device.

The attack described in the present paper aims at decrypting the correct VMK key which belongs to an encrypted memory unit through a dictionary attack to the user password or to the recovery password. That is, if an attacker is able to find the password to correctly decrypt the VMK key, she/he is able to decrypt the entire memory unit with that password.

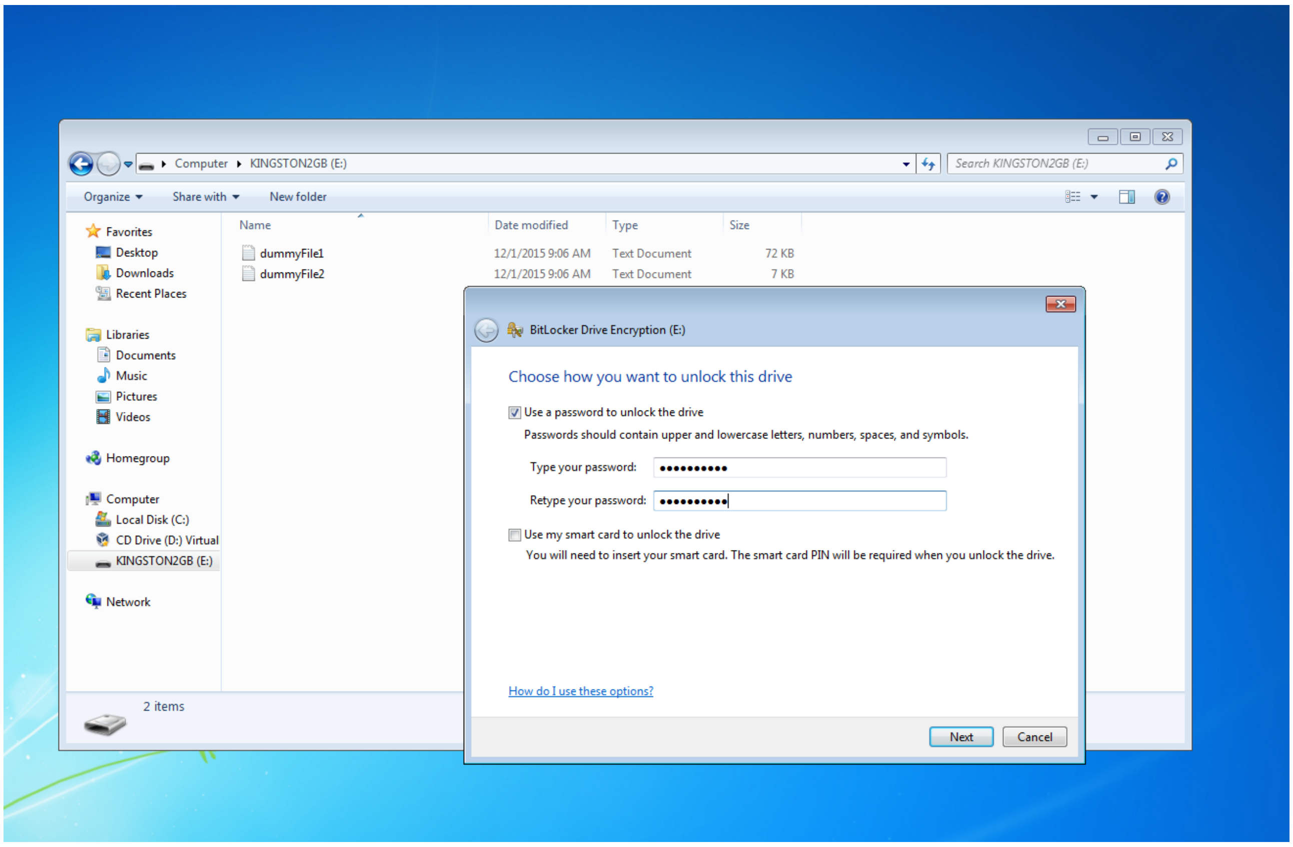
2.1 User Password VMK Decryption Procedure

To gain an insight about the workings of our attack, more information are necessary about the VMK decryption procedure (Figure 3) when the authentication method is a user password (see also (N. Kumar & V. Kumar, 2008) (Aorimn, 2018) and (Metz, 2018)):

1 https://blog.elcomsoft.com/2016/07/breaking-bitlocker-encryption-brute-forcing-the-backdoor-part-ii/

2 BitLocker To Go feature

3 Microsoft Blog: Recover Password method: https://docs.microsoft.com/en-us/windows/device-security/bitlocker/bitlocker-recovery-guide-plan



Name	Date modified	Type	Size
dummyFile1	12/1/2015 9:06 AM	Text Document	72 KB
dummyFile2	12/1/2015 9:06 AM	Text Document	7 KB

BitLocker Drive Encryption (E:)

Choose how you want to unlock this drive

☑ Use a password to unlock the drive

Passwords should contain upper and lowercase letters, numbers, spaces, and symbols.

Type your password: ••••••••••

Retype your password: ••••••••••

☐ Use my smart card to unlock the drive

You will need to insert your smart card. The smart card PIN will be required when you unlock the drive.

How do I use these options?

Figure 1: BitLocker encryption of an USB pendrive using the password authentication method.

1. the user provides the password;
2. SHA-256 is executed twice on it;
3. there is a loop of 0x100000 iterations, in which SHA-256 is applied to a structure like:

```
typedef struct {
    unsigned char updateHash[32];
        //last SHA-256 hash calculated
    unsigned char passwordHash[32];
        //hash from step 2
    unsigned char salt[16];
    uint64_t hash_count;
        // iteration number
} bitlockerMessage;
```

4. this loop produces an intermediate key, used with AES to encrypt the Initialization Vector (IV) (derived from a *nonce*);
5. XOR between encrypted IV and encrypted Message Authentication Code (MAC) to obtain the decrypted MAC;
6. XOR between encrypted IV and encrypted VMK to obtain the decrypted VMK;
7. if the MAC, calculated on the decrypted VMK, is equal to the decrypted MAC, the input password and the decrypted VMK are correct;

All the elements required by the decryption procedure (like VMK, MAC, IV, etc..) can be found inside the encrypted volume. In fact, during the encryption, BitLocker stores not only encrypted data but also metadata that provide information about encryption type, keys position, OS version, file system version and so on. Thanks to (Metz, 2018), (Aorimn, 2018), (N. Kumar & V. Kumar, 2008) and (Kornblum, 2009) we understood how to get all of these informations reading the BitLocker Drive Encryption (BDE) encrypted format. After an initial header, every BDE volume contains 3 (for backup purposes) FVE (Full Volume Encryption) metadata blocks, each one composed by a block header, a metadata header and an array of metadata entries.

In Figure 4 we report an example of FVE block belonging to a memory unit encrypted with Windows 8.1, enumerating the most interesting parts:

1. The "-FVE-FS-" signature, which marks the beginning of an FVE block
2. The Windows version number
3. The type and value of a VMK metadata entry
4. According to this value, the VMK has been encrypted using the user password authentication method
5. The salt of the VMK
6. According to this value, the type of VMK encryption is AES-CCM
7. Nonce
8. Message Authentication Code
9. Finally, the VMK

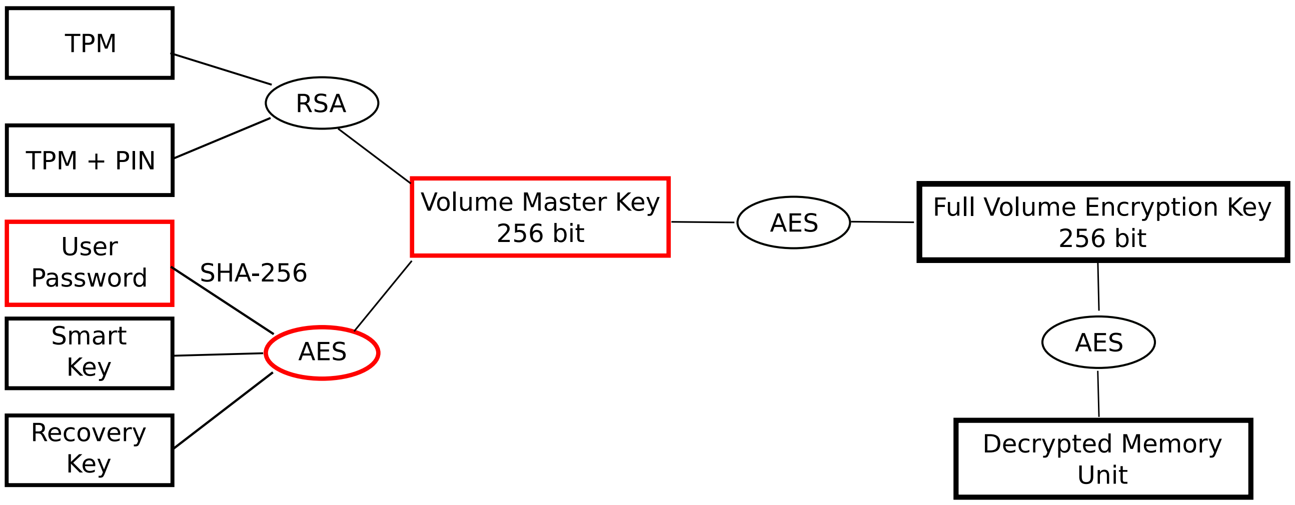


Figure 2: BitLocker encryption/decryption scheme

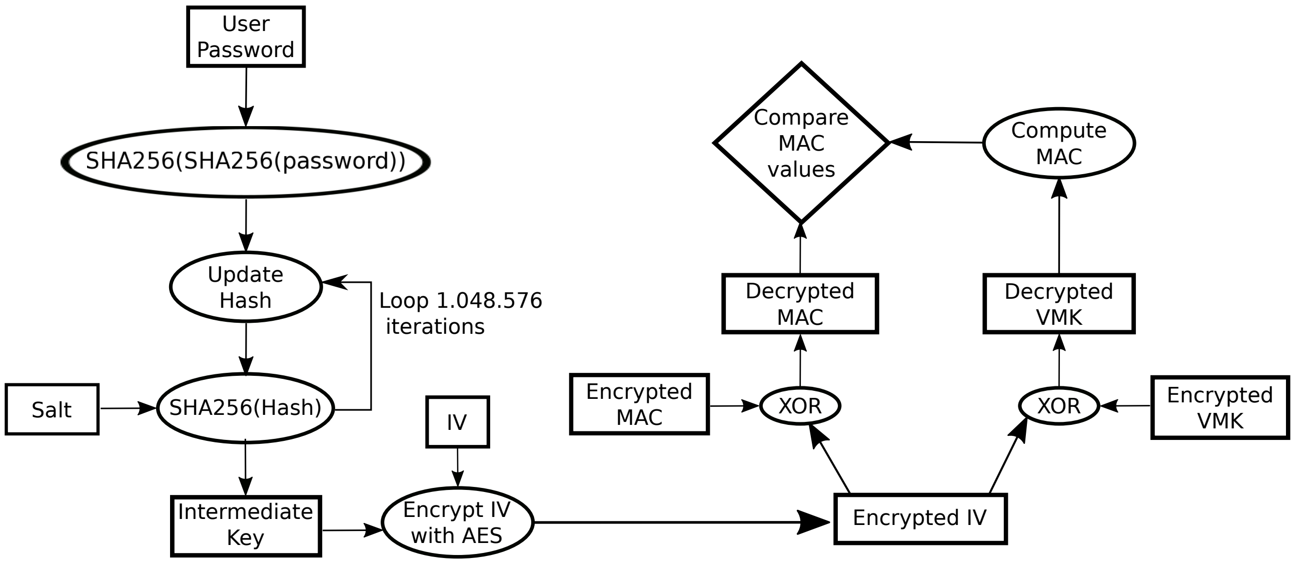


Figure 3: VMK decryption procedure

2.2 Recovery Password VMK Decryption Procedure

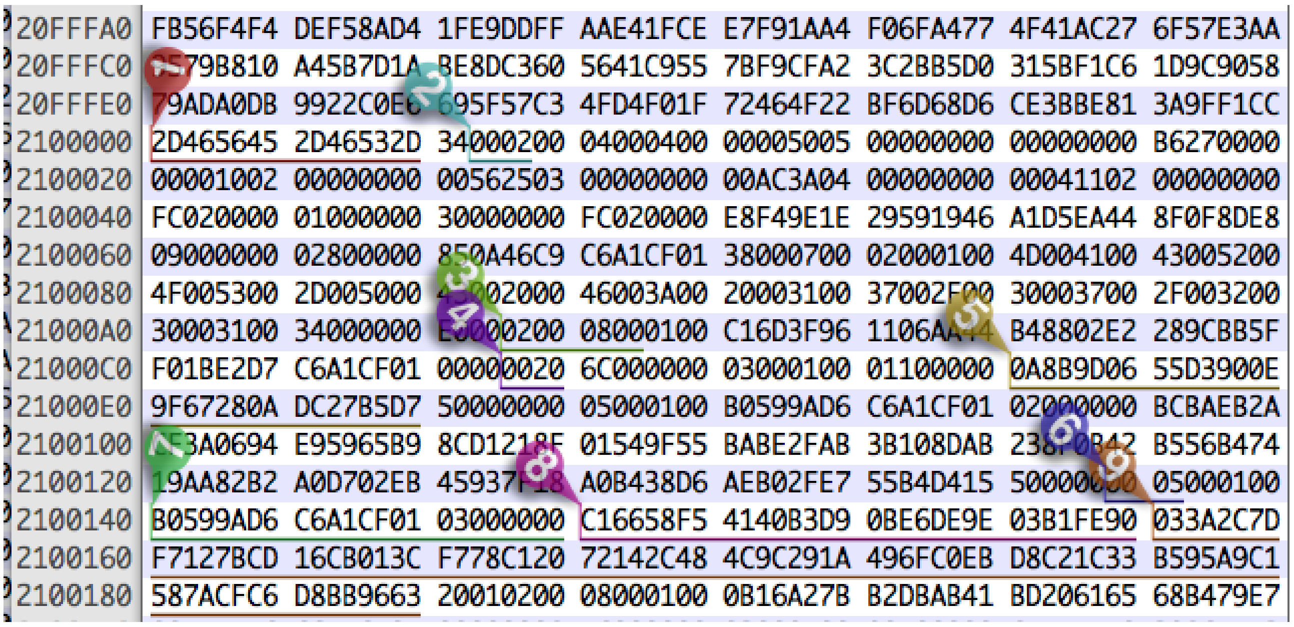
As above mentioned, the recovery password is a kind of passe-partout for all the authentication methods. According to (Kornblum, 2009), the recovery password is a 48-digit number composed by eight groups of six digits; each group of six digits must be divisible by eleven and must be less than 720896. Finally, the sixth digit in each group is a checksum digit. For instance, a valid recovery password is: 236808-089419-192665-495704-618299-073414-538373-542366. The number of all possible recovery password candidates is huge, thus building the entire dictionary would require too much storage.

The algorithm used by BitLocker to encrypt a device using the recovery password is similar to the user password one (with a few differences during the initial SHA-256 application): use the input password to produce an intermediate key useful to encrypt the VMK.

When the user encrypts a new memory device, regardless of the authentication method chosen, BitLocker always generates a recovery password; for this reason, every BitLocker encrypted memory unit has at least an encrypted VMK. Finally, performance in case of a recovery password attack is similar to the performance in case of a user password attack; therefore, during the rest of this paper, we report only about the performance of user password attacks.

3 BitCracker

Our software, named *BitCracker* (»BitCracker on GitHub«, 2018), aims at finding (starting from a dictionary) the key of a memory unit encrypted using the



```
20FFFA0  FB56F4F4 DEF58AD4 1FE9DDFF AAE41FCE E7F91AA4 F06FA477 4F41AC27 6F57E3AA
20FFFC0  [illegible]79B810 A45B7D1A BE8DC360 5641C955 7BF9CFA2 3C2BB5D0 315BF1C6 1D9C9058
20FFFE0  79ADA0DB 9922C0E[illegible] 695F57C3 4FD4F01F 72464F22 BF6D68D6 CE3BBE81 3A9FF1CC
2100000  2D465645 2D46532D 34000200 04000400 00005005 00000000 00000000 B6270000
2100020  00001002 00000000 00562503 00000000 00AC3A04 00000000 00041102 00000000
2100040  FC020000 01000000 30000000 FC020000 E8F49E1E 29591946 A1D5EA44 8F0F8DE8
2100060  09000000 02800000 850A46C9 C6A1CF01 38000700 02000100 4D004100 43005200
2100080  4F005300 2D005000 [illegible]002000 46003A00 20003100 37002F00 30003700 2F003200
21000A0  30003100 34000000 E0000200 08000100 C16D3F96 1106AA[illegible]4 B48802E2 289CBB5F
21000C0  F01BE2D7 C6A1CF01 00000020 6C000000 03000100 01100000 0A8B9D06 55D3900E
21000E0  9F67280A DC27B5D7 50000000 05000100 B0599AD6 C6A1CF01 02000000 BCBAEB2A
2100100  [illegible]3A0694 E95965B9 8CD1218F 01549F55 BABE2FAB 3B108DAB 238[illegible]0B42 B556B474
2100120  19AA82B2 A0D702EB 45937F18 A0B438D6 AEB02FE7 55B4D415 500000[illegible] 05000100
2100140  B0599AD6 C6A1CF01 03000000 C16658F5 4140B3D9 0BE6DE9E 03B1FE90 033A2C7D
2100160  F7127BCD 16CB013C F778C120 72142C48 4C9C291A 496FC0EB D8C21C33 B595A9C1
2100180  587ACFC6 D8BB9663 20010200 08000100 0B16A27B B2DBAB41 BD206165 68B479E7
```

Figure 4: FVE metadata block, BitLocker Windows 8.1

user password authentication or recovery password methods of BitLocker. It executes on GPUs (*Graphics Processing Units* [4]) the BitLocker decryption procedure with several performance improvements as described in the following sections:

- We introduced a preprocessing step before starting the main attack, to store in memory useful information for the SHA-256 based main loop (Section 3.1)
- We found a way to remove the final MAC computation and comparison (Section 3.2).

Finally, our code has been widely optimized for NVIDA GPUs (CUDA-C) but we implemented also an OpenCL version for portability reasons.

3.1 First improvement: SHA-256 and W Words

The most time-consuming part of the decryption algorithm is the loop of 0x100000 (1.048.576) SHA-256 operations, since a single hash involves many arithmetic operations. Moreover, during each iteration, the SHA-256 algorithm is applied twice to the 128 byte structure *bitlockerMessage* (Section 2.1) which is composed by several fields as shown in Table 1.

According to the SHA-256 standard (for a full description, see (of Standards & Technology, 2015)), the input message, before being hashed, is transformed into a set of so called *W blocks* according to the rule in Algorithm 1.

It is apparent that the first 16 W words depend on the original message and the others on the first 16 words. Therefore, looking at the message in Table 1 we were able to compute all possible W words useful for the SHA-256 of the second block of the message at each iteration in the loop, with no need to repeat many arithmetic operations during each iteration. Indeed, since for each encrypted memory unit, salt, padding and message size are always the same and `hash_count` is a number between 0 and (0x100000-1), we can precompute and store in memory all the W words, that require:

$$1.048.576 * 64 = 67.108.864 \; words * 4 \; byte \simeq 256Mb$$

This kind of improvement is specific for BitLocker (precomputation can be done if there is a part of the input message that is known ahead of time) and cannot be applied to a general SHA-256 implementation.

3.2 Second improvement: MAC comparison

During our analysis of the decrypted VMK's structure, using different Windows versions (7, 8.1 and 10) and a number of encrypted devices, we noticed several interesting facts:

1. The size of the VMK is always 44 bytes
2. First 12 bytes of decrypted VMK (Table 2) hold information about the key
   - First 2 bytes are the size of VMK, that is always 44 (0x002c)
   - Bytes 4 and 5 are the *version* number, always equal to 1
   - Byte 8 and 9 are the type of encryption. In case of user password, BitLocker always uses AES-CCM with a 256 bit key. So, ac-

4 https://it.wikipedia.org/wiki/Graphics_Processing_Unit

64-byte block #1		64-byte block #2			
32 bytes	32 bytes	16 bytes	8 bytes	32 bytes	8 bytes
updated_hash	password_hash	salt	hash_count	padding	message size
variable	fixed	fixed for each encrypted unit	between 0 and 0x100000	fixed	fixed to 88

Table 1: BitLocker SHA-256 message

Algorithm 1 SHA-256 standard algorithm, W blocks

```
1: Define $ROTR^n$(x) = $(x >> n) \vee (x << w - n)$ with $0 \leq n < w, w = 32$
2: Define $SHR^n$(x) = $(x >> n)$
3: Define $\sigma_0^{256} = ROTR^7(x) \oplus ROTR^{18}(x) \oplus SHR^3(x)$
4: Define $\sigma_1^{256} = ROTR^{17}(x) \oplus ROTR^{19}(x) \oplus SHR^{10}(x)$
5:
6: for $i$ = 1 to $N$ do
7:     Prepare the message schedule $W_t$
           $$W_t = \begin{cases} M_t^i & \text{if } 0 \leq t \leq 15 \\ \sigma_1^{256}(W_{t-2}) + W_{t-7} + \sigma_0^{256}(W_{t-15}) + W_{t-16} & \text{if } 16 \leq t \leq 63 \end{cases}$$
8:     applyHashFunction($W$)
9: end for
```

cording to the Microsoft standard, this value is between 0x2000 and 0x2005

3. Remaining 32 bytes are the key

Following these considerations, we removed the MAC test doing a simple check on the initial 12 bytes of the decrypted VMK, as shown in Figure 5.

To check the reliability of our solution, we tested BitCracker with several storage devices (both internal and USB-connected hard disks) encrypted by using passwords having between 8 and 16 characters under Windows 7 Enterprise Edition, Windows 7 Ultimate Edition and Windows 8 Pro N and Windows 10 Enterprise Edition (testing both BitLocker's compatible and non compatible modes) [5].

Although BitCracker always returned the correct output, some false positive may occur with this improved VMK check; for this reason BitCracker can be executed in 2 different modes: with (slower solution) or without (faster solution) the MAC comparison .

4 CUDA implementation performance

In this section we present the results of benchmarking activities of our stand-alone CUDA implementation of BitCracker with the improvements described in previous sections. We used several NVIDIA GPUs whose features are summarized in Table 3 [6].

During the following tests we always set the number of CUDA blocks to the maximum number of SM allowed by the GPU architecture: further increasing this number does not improve performance. The number of CUDA threads per block is always 1024 because each thread requires no more than 64 registers (we reached the maximum occupancy).

4.1 Kepler Architecture

We started to benchmark our final improved solution on the *Kepler* architecture using GPU GTK80 (Table 4).

The more the input grows, the better BitCracker performs. Increasing the number of blocks, each one with the same number of passwords per thread (*i.e.,* 8), leads to a better performance since the kernel launching overhead (that is basically constant) is distributed among more blocks.

4.2 Maxwell Architecture

In Table 5 we present the same benchmarks of the previous Section executed on the GFTX, using CC3.5 and CC5.2 (both available on the GPU).

It is worth to note that performance improves both due to the higher number of multiprocessors available in the new generation of NVIDIA cards and for the enhancements in integer instructions throughput [7]. This confirms that a well-tuned CUDA code can benefit from new features with a very limited effort.

5 Recently Microsoft introduced the BitLocker "Not Compatible" encryption mode in Windows 10: sectors of the memory device are encrypted with XTS-AES instead of AES-CCM. This change doesn't affect BitCrackers algorithm because there isn't any difference in the decryption procedure of the VMK.

6 *CC* is Compute Capability while *SM* is Stream Multiprocessors

7 NVIDIA Developer Zone Maxwell: https://developer.nvidia.com/maxwell-compute-architecture

Byte	0	1	2	3	4	5	6	7	8	9	10	11
Value	2c	00	00	00	01	00	00	00	03	20	00	00

Table 2: Example of initial 12 bytes of VMK decryption key

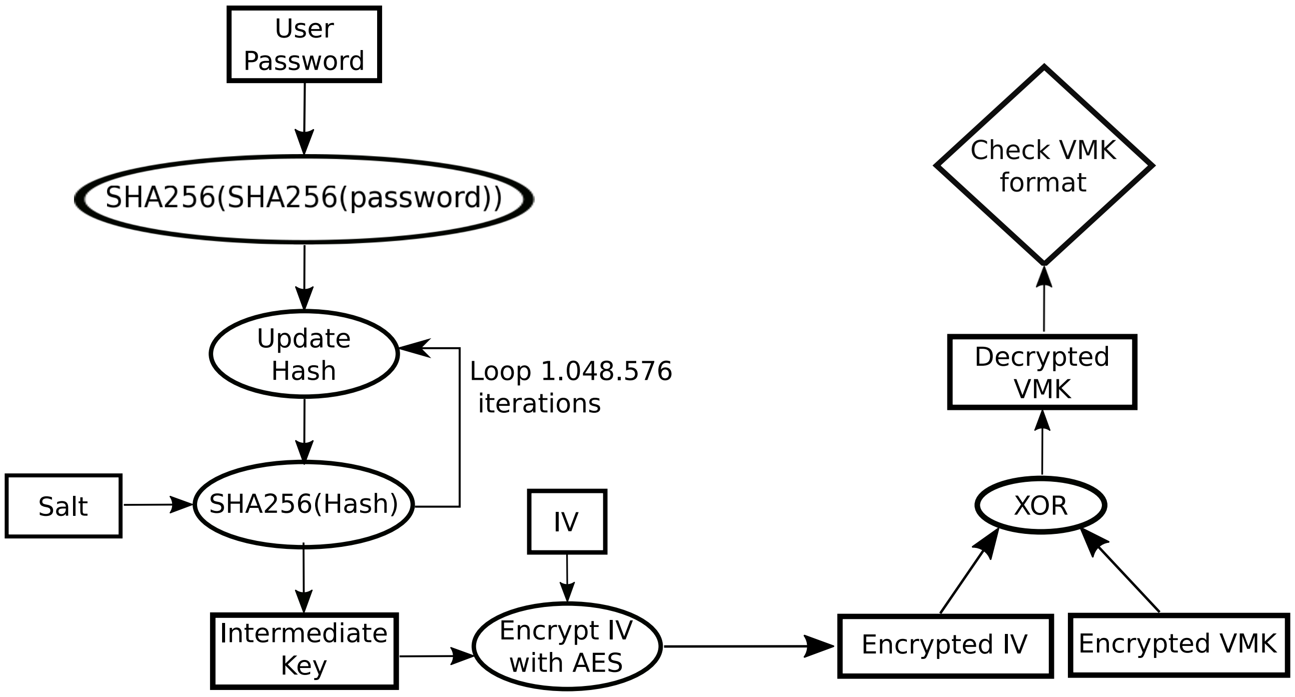


Figure 5: VMK decryption procedure improved

4.3 Pascal architecture

In Table 6, we summarize our benchmarks on GTP100. The performance improvement is close to a $\times$ 2 factor with respect to the Maxwell architecture, even if the main advantage of the new architecture (*i.e.*, the memory bandwidth that is about three times higher with respect to the *Kepler* architecture) has limited impact on a compute-intensive application like BitCracker.

5 OpenCL Implementation and John The Ripper

To make BitCracker available also to non-NVIDIA GPUs, we developed an OpenCL implementation. In order to take advantage of their system of *rules* for wordlist generation, our OpenCL implementation has been released also as a John the Ripper (Jumbo version) (»John the Ripper«, 2018) format (named *bitlocker-opencl*); the source code can be found here (»John the Ripper GitHub«, 2018) whereas the wiki reference page is here (»John the Ripper BitCracker Wiki Page«, 2018). When running *bitlocker-opencl* format, the John The Ripper internal engine autotunes all the OpenCL parameters (like local and global work groups). Running the following test, we reached up to 827 p/s passwords/second on the GTP100 .

6 Hash rate comparison

It is possible to evaluate BitCracker's performance by looking at the number of hashes per second that it computes (we recall that the check of each password requires 2.097.154 hashes, as described in section 2.1). The number of hashes *per* second that BitCracker is able to perform is summarized in Table 7[8] .

To assess BitCrackers performance, we carried out a comparison with the SHA-256 format (-m 1400) Hashcat (»Hashcat«, 2018) v4.1.0. We highlight that this is not a completely fair comparison since Hashcat does not execute exactly the same BitCracker's algorithm (BitCracker performs other operations beyond SHA-256) and it currently supports OpenCL only. The test in Listing 2 aims at providing an idea about the number of SHA256 that Hashcat is able to compute on our GTP100.

The resulting number of hashes per second is about 3290 MH/s that is comparable to BitCrackers best performance on the same GPU.

8 MH stands for MegaHashs

Listing 1: John The Ripper, BitLocker OpenCL format

```
$ ./john --format=bitlocker-opencl --mask=?a?a?a?a?a?a?a?a hash.txt
Device 0: Tesla P100-PCIE-16GB
Using default input encoding: UTF-8
Loaded 1 password hash (BitLocker-opencl, BitLocker [SHA256 AES OpenCL])
Cost 1 (iteration count) is 1048576 for all loaded hashes
Note: This format may emit false positives,
      so it will keep trying even after finding a possible candidate.
Note: minimum length forced to 8
0g 0:00:03:18  0g/s 795.8p/s 795.8c/s 795.8C/s GPU:47°C
     >Mdaaaaa..O7yaaaaa
0g 0:00:03:19  0g/s 827.8p/s 827.8c/s 827.8C/s GPU:47°C
     L7yaaaaa..;n5aaaaa
0g 0:00:03:20  0g/s 823.7p/s 823.7c/s 823.7C/s GPU:47°C
     L7yaaaaa..;n5aaaaa
0g 0:00:03:39  0g/s 817.7p/s 817.7c/s 817.7C/s GPU:47°C
     v;5aaaaa..\ 4aaaaa
0g 0:00:04:22  0g/s 820.2p/s 820.2c/s 820.2C/s GPU:47°C
     P)6aaaaa..1-baaaaa
```

Listing 2: Hashcat, SHA-256 format

```
./hashcat -m 1400 -a 3 -d 3 -O -w 3 hash.txt  ?a?a?a?a?a?a?a?a
....
Session........: hashcat
Status.........: Running
Hash.Type......: SHA-256
Hash.Target....: 68585251d17afaec3d0dd2f5315ee5a826a708d3c94f ... 97aa69
Time.Started...: Fri Jun  8 15:54:03 2018 (1 min, 11 secs)
Time.Estimated.: Mon Jul  2 00:18:53 2018 (23 days, 8 hours)
Guess.Mask.....: ?a?a?a?a?a?a?a?a [8]
Guess.Queue....: 1/1 (100.00%)
Speed.Dev.#3...:  3288.3 MH/s (71.07ms) Accel:32 Loops:128 Thr:1024 Vec:1
Recovered......: 0/1 (0.00%) Digests, 0/1 (0.00%) Salts
Progress.......: 234646142976/6634204312890625 (0.00%)
Rejected.......: 0/234646142976 (0.00%)
Restore.Point..: 0/7737809375 (0.00%)
Candidates.#3..: 1p2erane -> SC2[7sta
HWMon.Dev.#3...: Temp: 51c Util:100% Core:1328MHz Mem: 715MHz Bus:16
```

Acronim	Name	Arch	CC	# SM	CUDA
GTK80	Tesla K80	Kepler	3.5	13	7.0/7.5
GFTX	GeForce Titan X	Maxwell	5.2	24	7.5
GTP100	Tesla P100	Pascal	6.1	56	8.0

Table 3: NVIDIA GPUs used for bench

Blocks	Threads/Block	Pwds/Thread	Pwds/Kernel	Seconds	Pwds/Sec
1	1.024	1	1.024	30	33
1	1.024	8	8.192	245	33
2	1.024	8	16.384	247	66
4	1.024	8	32.768	248	132
8	1.024	8	65.536	253	258
13	1.024	8	106.496	276	385

Table 4: GTK80 benchmarks

7 Conclusions

We presented the first open-source implementation of a tool for efficient dictionary attacks to the BitLocker crypto system.
The results show that our BitCracker may compete with a *state-of-the art* password cracker in terms of raw performance on the basic computational kernels whilst it is the only one providing specific shortcuts to speedup the BitLocker decryption procedure. We can conclude that, although the complex architecture of BitLocker reduces significantly the number of passwords that is possible to test in a unit of time, with respect to other crypto-systems (*e.g.,* OpenPGP), it is still necessary to pay special attention to the choice of the user password, since, with a single high-end GPU, more than a quarter-billion of passwords can be tested in a day (~ 1418 passwords *per* second on a GTP100 $\times\ 86400$ seconds $\simeq 122$ million in a day). Our implementations of SHA-256, fully customized for the CUDA-C environment, can be reused (provided that the W words optimization is turned off, since it cannot be applied to a general situation) for any procedure that requires to use that hash function (*e.g.*, HMAC-SHA256).

Other possible improvements include the enhancement of BitCracker by adding a mask mode attack and/or a smart reading of the input dictionary (*e.g.* by assigning a probability to them) that are available in most widely used password crackers.

We released our CUDA and OpenCL standalone implementations on GitHub here (»BitCracker on GitHub«, 2018) and as *bitlocker-opencl* format for John The Ripper (»John the Ripper«, 2018).

About the Authors

Elena Agostini received her Ph.D. in Computer Science from the University of Rome »Sapienza« in collaboration with the National Research Council of Italy. Her main scientific interests are parallel computing, GPGPUs, HPC, network protocols and cryptanalysis.

Massimo Bernaschi has been 10 years with IBM working in High Performance Computing. Currently he is with the National Research Council of Italy (CNR) as Chief Technology Officer of the Institute for Computing Applications. He is also an adjunct professor of Computer Science at "Sapienza" University in Rome. He has been named CUDA Fellow in 2012.

References

Aorimn. (2018). Dislocker: Fuse driver to read/write windows' bitlocker-ed volumes under linux/mac osx. Retrieved from https://github.com/Aorimn/dislocker

BitCracker on GitHub. (2018). Retrieved from https://github.com/e-ago/bitcracker

Elcomsoft Forensic Disk Decryptor. (2018). Retrieved from https://www.elcomsoft.com/efdd.html

Hashcat. (2018). Retrieved from https://hashcat.net/hashcat

John the Ripper. (2018). Retrieved from http://www.openwall.com/john

John the Ripper BitCracker Wiki Page. (2018). Retrieved from http://openwall.info/wiki/john/OpenCL-BitLocker

John the Ripper GitHub. (2018). Retrieved from https://github.com/magnumripper/JohnTheRipper

Kornblum, J. D. (2009). Implementing bitlocker drive encryption for forensic analysis. *Digital Investigation: The International Journal of Digital Forensics & Incident Response*, *5*, 75–84.

Metz, J. (2018). Bitlocker drive encryption (bde) format specification. Retrieved from https://github.com/libyal/libbde/tree/master/documentation

N. Kumar & V. Kumar. (2008). Bitlocker and windows vista. Retrieved from http://www.nvlabs.in/uploads/projects/nvbit/nvbit_bitlocker_white_paper.pdf

Secure Hash Standard (SHS). (2015). Retrieved from http://dx.doi.org/10.6028/NIST.FIPS.180-4

CC	Blocks	Threads/Block	Pwds/Thread	Pwds/Kernel	Seconds	Pwds/Sec
3.5	1	1.024	1	1.024	24	42
3.5	1	1.024	8	8.192	191	42
3.5	24	1.024	8	196.608	212	925
3.5	24	1.024	128	3.145.728	3496	900
5.2	1	1.024	1	1.024	23	44
5.2	1	1.024	8	8.192	188	43
5.2	24	1.024	8	196.608	210	933
5.2	24	1.024	128	3.145.728	3369	933

Table 5: GFTX benchmarks, CC3.5 and CC5.2

CC	Blocks	Threads/Block	Pwds/Thread	Pwds/Kernel	Seconds	Pwds/Sec
6.1	1	1.024	1	1.024	38	26
6.1	56	1.024	1	57.344	40	1.418
6.1	56	1.024	8	458.752	336	1.363
6.1	56	1.024	128	7.340.032	5444	1.348

Table 6: GTP100 benchmark

GPU	Password/Sec	Hash/Sec
GTK80	385	807 MH/s
GFTX	933	1.957 MH/s
GTP100	1.418	2.973 MH/s

Table 7: BitCracker's hashes per second, CUDA implementation