Decrypting Bit Locked drive: Using Open-Source Tools

Ashmit Sharma[1], Anju Anna Joseph[2], Kanan Bala Jena[3]
[1]*Scientist 'B', Forensic Electronics, CFSL, Bhopal Camp at CFSL, Chandigarh*
[2]*Forensic Professional, Ballistics, CFSL, Bhopal*
[3]*Director & Scientist 'E', CFSL, Bhopal*

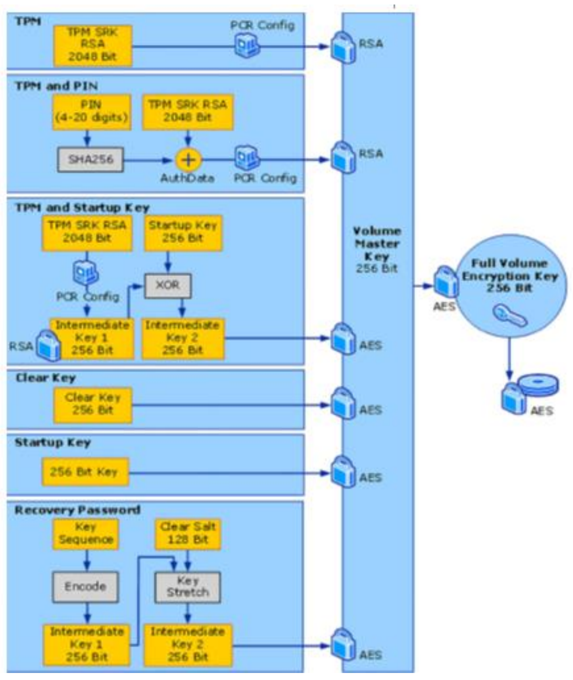
***Abstract*— BitLocker encryption is designed to enhance data security by encrypting the entire hard drive, unlike traditional systems that only provide partial encryption. While this offers robust protection, it presents a significant challenge for forensic investigators, as all data on the drive becomes inaccessible without decryption. Forensic access to BitLocker-encrypted volumes relies on obtaining key protectors such as the Volume Master Key (VMK) or the Full Volume Encryption Key (FVEK). The use of the VMK as an intermediate key enables the modification of compromised protectors without re-encrypting the drive's data. This study explores the feasibility of decrypting BitLocker-encrypted volumes using open-source tools and assesses their effectiveness. The research aims to contribute to the development of forensic methodologies for encrypted data access. Furthermore, while commercial decryption tools employing similar techniques often come with significant costs, this study highlights how open-source alternatives provide cost-effective solutions to break the Bitlocked encrypted drive. Hence, offering a valuable resource for investigators with limited budgets.**

***Index Terms*—Breaking Bit-locked drives, Encryption, Decryption of Bit-locked drives, Open source forensics, Digital Forensics, brute force attack.**

I. INTRODUCTION

BitLocker is a Windows security feature that provides encryption for entire volumes, addressing the risks of data theft or exposure from lost, stolen, or improperly decommissioned devices. It offers maximum protection when used with a Trusted Platform Module (TPM), a common hardware component in Windows devices. The TPM works with BitLocker to ensure that the device has not been tampered with while the system is offline. In addition to using a TPM, BitLocker can secure the normal startup process by requiring the user to enter a personal identification number (PIN) or insert a removable device containing a startup key. These security measures provide multifactor authentication and ensure that the device cannot start or resume from hibernation until the correct PIN or startup key is presented. For devices without a TPM, BitLocker can still be used to encrypt the operating system drive. BitLocker's two main features are providing full drive encryption and verifying the integrity of early boot components.

A. BITLOCKER ARCHITECTURE



B. Full Disk Encryption (FDE)

It encrypts every file folder, sector with strong cryptographic algorithm. Bit locker of Windows 10 supports XTS-AES (Advanced Encryption Standard) which provides protection against unauthorized manipulation of encrypted data. There are two types of encryption namely software based and hardware based encryption. The former encrypts the operating system and other data partitions whereas the latter

uses user entered PIN or Trusted Computing Platform (TPM) in addition to encrypting the operating system. The contents of the drive are challenging for an adversary to access, as they are implemented within embedded hardware chips.
In software based encryption, the keys used for encryption are stored in the system memory which can be accessed making it vulnerable for certain attacks such as cold boot attacks. After the booting process, since the keys are located in the RAM which makes it feasible to circumvent the encryption.

C. Self-Encryption Drives (SDE)

These are hard wares with inbuilt hardware-based encryption modules. All encryption keys are stored natively on the hard drive and the keys remain confined to the hard disk and hence not accessible by any software running over the operating system. SED generates two types of keys: Media Encryption Key (MEK) & Key Encryption Key (KEK).
MEK: used for encrypting the contents of hard drive
KEK: used for encrypting MEK.

II. INSTRUMENTATION

A. High End Workstation

A high-end workstation is required for a BitLocker dictionary attack because of the significant computational power needed to handle the complexity of the encryption and the large size of the dictionary. BitLocker uses strong encryption (AES with 128-bit or 256-bit keys), creating a vast search space for password combinations. A dictionary attack tests millions or even billions of potential passwords, which requires massive processing power. High-end workstations equipped with powerful CPUs and GPUs enable parallel processing, significantly speeding up the testing of these combinations. Additionally, large dictionaries and the need for substantial memory and storage capacity make powerful hardware essential to efficiently process and crack the encryption, especially for complex passwords. Without such high-performance equipment, the attack would be too slow and impractical, even taking months or longer for successful decryption.

B. FTK ImagerFTK Imager is a tool primarily used for creating forensic disk images, and while it does not directly support decrypting BitLocker encryption, it can play a role in the process. First, FTK Imager can be used to create an exact image of the encrypted disk, which is crucial for forensic analysis. After imaging the drive, forensic investigators can attempt a password recovery using specialized tools. FTK Imager can also be used to analyze the disk image for any clues, such as recovery keys or encrypted data remnants. However, the actual decryption or password attack requires additional tools, as FTK Imager itself does not have the capability to crack passwords or bypass encryption.
Download Link: https://www.exterro.com/ftk-product-downloads/ftk-imager-4-7-3-81

C. Hash Cat

Hashcat is a powerful, open-source password cracking tool that can be used to decrypt BitLocker-encrypted drives by attacking the password or PIN used as a key protector. BitLocker encryption stores password hashes, and Hashcat can leverage these hashes to attempt to recover the password through methods like dictionary, brute-force, or hybrid attacks. By utilizing advanced algorithms and hardware acceleration, particularly through GPUs**,** Hashcat can test millions of password combinations per second, drastically speeding up the cracking process. The effectiveness of Hashcat depends on the complexity of the password, with simple passwords being more easily cracked, while longer and more complex passwords requiring substantial computational power and time. Hashcat's ability to handle large datasets and perform parallel computations makes it a powerful tool for forensic investigators and attackers attempting to bypass BitLocker encryption.
Download Link: https://hashcat.net/hashcat/

D. JOHN THE RIPPER

John the Ripper is another popular open-source password cracking tool that can be used to decrypt BitLocker-encrypted drives. Similar to Hashcat, John the Ripper is designed to break password hashes through various attack methods like dictionary, brute-force, and hybrid attacks. For BitLocker decryption, John the Ripper works by targeting the BitLocker password hash stored in the encrypted volume. By extracting the hash from the BitLocker-encrypted drive, John the Ripper can attempt to crack the

password that protects the encryption key, allowing access to the encrypted data.
John the Ripper utilizes advanced techniques and can be customized with different wordlists or rules to improve its chances of success, making it adaptable to a wide range of password structures. However, its performance is typically slower than tools like Hashcat, particularly when cracking passwords with complex combinations, as it doesn't utilize GPUs for acceleration. Despite this, John the Ripper remains a valuable tool for forensic investigators and security professionals, especially when dealing with simpler or more predictable passwords. It can be used in combination with other tools to enhance the overall decryption process when trying to access data on a BitLocker-encrypted drive.
Download Link: https://www.openwall.com/john/

E. Password Dictionary

In a brute force BitLocker decryption attack, the use of a password dictionary significantly enhances the efficiency of the process by narrowing the search space. Rather than attempting every possible character combination, which can be incredibly slow, a dictionary contains likely passwords, including common words, phrases, and predictable patterns that many users tend to choose. This reduces the number of combinations that need to be tested, speeding up the attack. Additionally, dictionaries can be customized based on personal information or known details about the user, further improving success rates. A hybrid approach combining brute force and dictionary attacks is often employed, where the dictionary tests common passwords and brute force handles variations or more complex combinations. This approach is especially effective against weak passwords, which are common in BitLocker encryption, making the process of decryption more efficient and less time-consuming. Such dictionaries are available on the internet and can be downloaded.
Download Link: https://github.com/zxcV32/indian-wordlist

III. OVERVIEW

This research explores the use of open-source password cracking tools like Hashcat and John the Ripper for decrypting BitLocker-encrypted devices. The focus of the study is to assess the effectiveness of these tools in attacking BitLocker encryption and extracting password hashes or decrypting encrypted data, both from software-based encryption (such as BitLocker) and hardware-based encryption (including Self-Encrypting Drives, or SEDs).
The primary objective of this research is to evaluate the feasibility and efficiency of these open-source tools in forensic investigations**.** Specifically, we examine their ability to recover passwords from encrypted drives, which can be critical in legal and forensic contexts. We also investigate the performance of Hashcat and John the Ripper in password cracking using dictionary-based and brute-force attack methods.
By focusing on real-world scenarios, the research aims to determine how these tools can assist digital forensic professionals in accessing encrypted data when traditional decryption methods are unavailable or ineffective. Additionally, the study considers the legal and ethical implications of using these tools in forensic investigations, emphasizing the importance of proper authorization and lawful use.
Ultimately, this research contributes to the growing field of digital forensics, highlighting the potential and limitations of open-source tools like Hashcat and John the Ripper in decrypting BitLocker-encrypted volumes and aiding in forensic evidence recovery.

IV. LITERATURE REVIEW

A. BitLocker Encryption

BitLocker uses a combination of AES (Advanced Encryption Standard) and PBKDF2 (Password-Based Key Derivation Function 2) to protect the drive data. The key steps are:

- PBKDF2 is used to derive the encryption key (used for encrypting the Volume Master Key (VMK)) from the password.
- The VMK is then encrypted with AES to protect the drive's contents.
- The encrypted VMK is stored in the BitLocker header.
- Formula for Key Derivation (PBKDF2)
- BitLocker uses PBKDF2 (Password-Based Key Derivation Function 2) to convert the user’s password into the key that is used to encrypt and protect the Volume Master Key (VMK).

$$\text{Derived Key} = \text{HMAC-SHA1}(P, S, I)$$

Where:

- PPP = Password or passphrase used by the user to unlock the drive.
- SSS = Salt, which is a random value added to the password to ensure uniqueness (different salts are used in each encryption).
- III = The iteration count, which specifies how many times the HMAC-SHA1 function is applied to slow down brute-force attacks. In BitLocker, this iteration count is typically 100,000 or higher.

  This process results in a derived key that is then used to decrypt the Volume Master Key (VMK), which is itself encrypted with AES.

B. Bitlocker2john

Bitlocker2john is a tool included in the John the Ripper suite designed to extract password hashes from BitLocker-encrypted volumes. To extract the hash used in the cracking process, BitLocker2John performs the following steps:

Step 1: Extract BitLocker Header

The tool first reads the BitLocker header from the disk, which contains the encrypted VMK, salt, and iteration count used in the PBKDF2 process.

Step 2: Apply PBKDF2

The password is hashed using PBKDF2 to derive the key that will be used to decrypt the VMK. This is done by applying HMAC-SHA1 on the password and salt for a number of iterations.

$$\text{Derived Key} = \text{PBKDF2-HMAC-SHA1}(P, S, \text{Iterations})$$

Where:

- P is the user password,
- S is the salt value,
- Iterations is the number of PBKDF2 iterations

C. Hashcat and GPU-Based Cracking

Hashcat is a powerful password cracking tool that leverages GPU acceleration to efficiently break hash-based password protections. It supports a broad spectrum of cryptographic hash algorithms (e.g., MD5, SHA-1, bcrypt) and employs various attack modes such as brute-force, dictionary, mask, and hybrid attacks. The mathematics behind Hashcat involves a combination of cryptographic hash functions, key derivation functions (KDFs), and strategic attack methods that exploit the computational power of GPUs to crack passwords.

1. Cryptographic Hash Functions

At the heart of Hashcat's functionality is its ability to crack cryptographic hash functions used to securely store passwords. A hash function takes an input (or "message") and produces a fixed-length output, known as the "hash." The essential properties of a cryptographic hash function are:

- Deterministic: The same input will always produce the same hash.
- Fast to compute: Hash functions are designed to be computed quickly.
- Pre-image resistance: It is computationally challenging to reverse the hash function (i.e., to find an input corresponding to a given hash).
- Collision resistance: It is difficult to find two different inputs that produce the same hash value.
- Mathematically, a hash function H maps an input message m to a fixed-size output h:

$$h = H(m)$$

Where:

- m is the input message (e.g., a password).
- h is the resulting hash value.
- These properties ensure that a hash function is useful for securely storing passwords, but also present challenges for attackers trying to reverse-engineer the original input.

2. Key Derivation Functions (KDFs)

In addition to simple hash functions, Hashcat also targets Key Derivation Functions (KDFs), such as PBKDF2 and bcrypt**,** which are designed to make the password hashing process slower and more resistant to brute-force attacks.

2.1 PBKDF2 (Password-Based Key Derivation Function 2)

PBKDF2 is widely used in password hashing schemes to protect against brute-force attacks. It applies a hash function (e.g., SHA-256) to the password in multiple iterations to derive a key. The formula for PBKDF2 is:

$$DK = \mathrm{HMAC}(P, S, i)$$

Where:

- DK is the derived key (output).
- P is the password (input).
- S is the salt (random data to prevent identical passwords from generating the same derived key).
- i is the iteration count (number of hash rounds).
- HMAC is the hash-based message authentication code, typically computed using SHA-1 or SHA-256.
- The number of iterations iii is typically large (e.g., 100,000 or more) to slow down brute-force attempts, making password cracking significantly harder.

2.2 bcrypt

bcrypt is another widely used KDF for password hashing that incorporates the Blowfish encryption algorithm. It applies the Blowfish algorithm in a way that increases the computational cost, making password cracking more difficult. bcrypt uses a salt and a cost factor ccc, which determines the number of rounds of hashing applied. The formula for bcrypt is:

$$\mathrm{bcrypt}(P, S, c) = \mathrm{Blowfish}(S, P) \text{ repeated } c \text{ times}$$

Where:

- P is the password.
- S is the salt.
- c is the cost factor (the number of iterations).
- As the cost factor ccc increases, the time required to compute the hash also increases, which slows down brute-force attacks.

D. Attack Modes in Hashcat

Hashcat supports various attack modes, each employing different mathematical strategies to guess the correct password. The most common attack modes are:

1 Dictionary Attack

In a dictionary attack, Hashcat attempts passwords from a predefined list (dictionary) of commonly used passwords. The time complexity of this attack is proportional to the size of the dictionary, represented as:

T = N

Where N is the number of entries in the dictionary. Dictionary attacks are faster than brute-force attacks since they leverage commonly used password patterns.

E. GPU Acceleration in Hashcat

One of Hashcat's key strengths is its use of GPU acceleration. GPUs are designed for parallel computation and are highly effective at performing many hash calculations simultaneously. This parallelism allows Hashcat to dramatically increase the number of hashes it can compute per second compared to traditional CPU-based cracking.

The number of hashes per second can be represented as:

$$\text{Hashes per second} = H \times P$$

Where:

- H is the number of hashes that can be computed in parallel per cycle (dependent on the algorithm and the GPU architecture).
- P is the number of parallel operations that can be performed (based on the number of GPU cores).
- This parallel computing capability allows Hashcat to perform password cracking far more efficiently, significantly reducing the time required to attempt a large number of possible passwords.

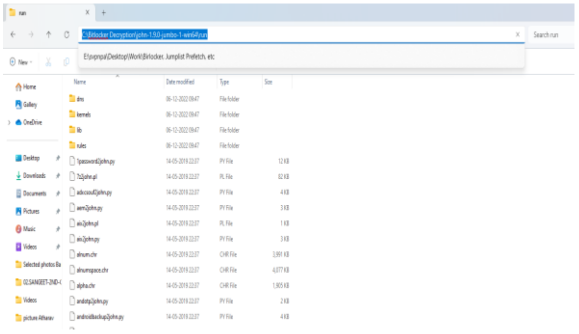
V. METHODOLOGY

A. Create Forensic Image of the Encrypted Drive

To create a forensic image of an encrypted drive using FTK Imager, first launch the tool and select Create Disk Image from the File menu. Next, choose the encrypted drive (either physical or logical) as the source. Select a destination folder to save the image, and choose the preferred image format (e.g., E01, Raw dd**)**. Configure the settings to generate hash values (MD5, SHA1, SHA256) for integrity verification and enable compression if needed. Click Start to begin the imaging process, ensuring the drive remains unaltered during the process. Once the image is created, verify the hashes to confirm the image matches the original drive, and securely store the image file while maintaining proper chain of custody documentation. This ensures a bit-for-bit copy of the encrypted drive, preserving its integrity, though the encrypted data will need decryption methods like password cracking to access its contents.

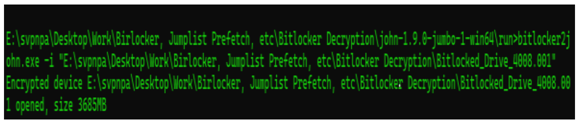
B. Bitlocker2John

Navigate to the "Run" folder within the John the Ripper tool, type CMD in the highlighted area, and press Enter.


run
C:\Bitlocker Decryption\john-1.9.0-jumbo-1-win64\run
E:\svpnpa\Desktop\Work\Birlocker, Jumplist Prefetch, etc
Search run
Home
Gallery
OneDrive
Desktop
Downloads
Documents
Pictures
Music
Videos
Name
Date modified
Type
Size

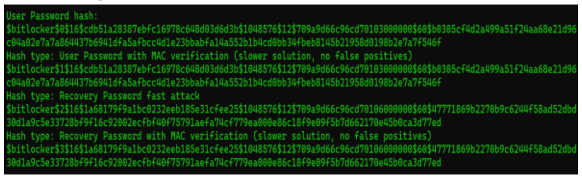
Type bitlocker2john.exe -i "location of the forensic image" in the command prompt for eg. bitlocker2john.exe -i E:\Bitlocker Decryption\Bitlocked.E01"

bitlocker2john.exe extracts the hash value(s) associated with the BitLocker encryption from the provided drive image file. The hash is needed to perform the password cracking process



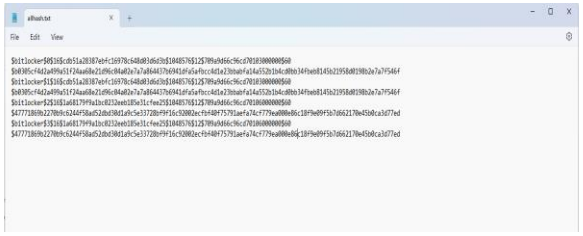
```
E:\svpnpa\Desktop\Work\Birlocker, Jumplist Prefetch, etc\Bitlocker Decryption\john-1.9.0-jumbo-1-win64\run>bitlocker2j
ohn.exe -i "E:\svpnpa\Desktop\Work\Birlocker, Jumplist Prefetch, etc\Bitlocker Decryption\Bitlocked_Drive_4008.001"
Encrypted device E:\svpnpa\Desktop\Work\Birlocker, Jumplist Prefetch, etc\Bitlocker Decryption\Bitlocked_Drive_4008.00
1 opened, size 3685MB
```

Scroll down, and you will find the password hashes displayed. The tool will output the hash values in the following format:



```
User Password hash:
$bitlocker$0$16$cdb51a28387ebfc16978c648d03d6d3b$1048576$12$709a9d66c96cd70103000000$60$b0305cf4d2a499a51f24aa68e21d96
c04a02e7a7a864437b6941dfa5afbcc4d1e23bbabfa14a552b1b4cd0bb34fbeb8145b21958d0198b2e7a7f546f
Hash type: User Password with MAC verification (slower solution, no false positives)
$bitlocker$1$16$cdb51a28387ebfc16978c648d03d6d3b$1048576$12$709a9d66c96cd70103000000$60$b0305cf4d2a499a51f24aa68e21d96
c04a02e7a7a864437b6941dfa5afbcc4d1e23bbabfa14a552b1b4cd0bb34fbeb8145b21958d0198b2e7a7f546f
Hash type: Recovery Password fast attack
$bitlocker$2$16$1a68179f9a1bc0232eeb185e31cfee25$1048576$12$709a9d66c96cd70106000000$60$47771869b2270b9c6244f58ad52dbd
30d1a9c5e33728bf9f16c92002ecfbf40f75791aefa74cf779ea000e86c18f9e09f5b7d662170e45b0ca3d77ed
Hash type: Recovery Password with MAC verification (slower solution, no false positives)
$bitlocker$3$16$1a68179f9a1bc0232eeb185e31cfee25$1048576$12$709a9d66c96cd70106000000$60$47771869b2270b9c6244f58ad52dbd
30d1a9c5e33728bf9f16c92002ecfbf40f75791aefa74cf779ea000e86c18f9e09f5b7d662170e45b0ca3d77ed
```

Copy these hash values and paste them into a Notepad file, then save it as **allhash.txt**.



```
$bitlocker$0$16$cdb51a28387ebfc16978c648d03d6d3b$1048576$12$709a9d66c96cd70103000000$60
$b0305cf4d2a499a51f24aa68e21d96c04a02e7a7a864437b6941dfa5afbcc4d1e23bbabfa14a552b1b4cd0bb34fbeb8145b21958d0198b2e7a7f546f
$bitlocker$1$16$cdb51a28387ebfc16978c648d03d6d3b$1048576$12$709a9d66c96cd70103000000$60
$b0305cf4d2a499a51f24aa68e21d96c04a02e7a7a864437b6941dfa5afbcc4d1e23bbabfa14a552b1b4cd0bb34fbeb8145b21958d0198b2e7a7f546f
$bitlocker$2$16$1a68179f9a1bc0232eeb185e31cfee25$1048576$12$709a9d66c96cd70106000000$60
$47771869b2270b9c6244f58ad52dbd30d1a9c5e33728bf9f16c92002ecfbf40f75791aefa74cf779ea000e86c18f9e09f5b7d662170e45b0ca3d77ed
$bitlocker$3$16$1a68179f9a1bc0232eeb185e31cfee25$1048576$12$709a9d66c96cd70106000000$60
$47771869b2270b9c6244f58ad52dbd30d1a9c5e33728bf9f16c92002ecfbf40f75791aefa74cf779ea000e86c18f9e09f5b7d662170e45b0ca3d77ed
```

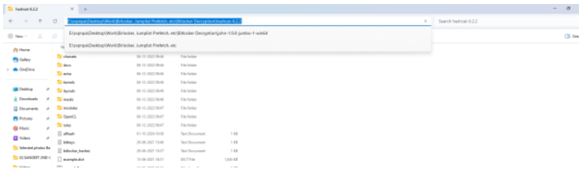
C. Hashcat

Once you have copied the hash values and saved them as allhash.txt, move this file, along with the password dictionary file (e.g., passes.txt), into the Hashcat folder for easier access during the cracking process.

The dictionary file (passes.txt) contains a list of potential passwords that might match the BitLocker password. This file is essential for performing a dictionary attack.The passes.txt file should contain one password per line. This file may contain common passwords, phrases, or other potential passwords relevant to the target system.

A large and diverse dictionary file increases the chances of successfully cracking the password.

Navigate to the Hash Cat tool, type CMD in the highlighted area, and press Enter.



Type hashcat.exe -m 22100 hash.txt passes.txt –show in CMD prompt

hashcat.exe: This is the executable for Hashcat, a powerful password cracking tool.

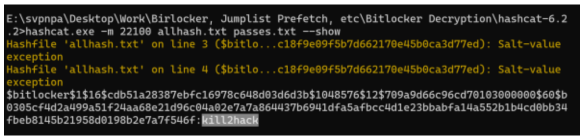
-m 22100: This specifies the hash mode for BitLocker encryption. Mode 22100 corresponds to the BitLocker hash.

hash.txt: The file containing the BitLocker hash value extracted in

passes.txt: The dictionary file containing possible passwords to try against the hash.

--show: This flag tells Hashcat to display the cracked password if it successfully matches the hash.

Hashcat will begin processing the hash and attempting to match it with each password in passes.txt.
If it finds a match, it will display the cracked password, for example: kill2hack is the decryption key in our case



```
E:\svpnpa\Desktop\Work\Birlocker, Jumplist Prefetch, etc\Bitlocker Decryption\hashcat-6.2
.2>hashcat.exe -m 22100 allhash.txt passes.txt --show
Hashfile 'allhash.txt' on line 3 ($bitlo...c18f9e09f5b7d662170e45b0ca3d77ed): Salt-value
exception
Hashfile 'allhash.txt' on line 4 ($bitlo...c18f9e09f5b7d662170e45b0ca3d77ed): Salt-value
exception
$bitlocker$1$16$cdb51a28387ebfc16978c648d03d6d3b$1048576$12$709a9d66c96cd70103000000$60$b
0305cf4d2a499a51f24aa68e21d96c04a02e7a7a864437b6941dfa5afbcc4d1e23bbabfa14a552b1b4cd0bb34
fbeb8145b21958d0198b2e7a7f546f:kill2hack
```

VI. OUTCOME

The analysis of this experiment on cracking BitLocker-encrypted drives using dictionary attacks offers valuable insights into the strengths, limitations, and key factors influencing the success of such methods. Several aspects play a critical role in determining the outcome, which are explored in more detail below:

A. Quality and Size of the Dictionary

A primary factor in the success of dictionary attacks is the quality and size of the dictionary used. The passes.txt file, which serves as the list of potential passwords, must be comprehensive enough to account for common patterns, variations, and complex password choices. For common passwords like "123456" or "password," a basic dictionary file might be sufficient. However, as Bonneau (2012) suggests, for complex passwords involving multiple characters, symbols, and numbers, a more extensive and specialized dictionary would be necessary. A smaller or outdated dictionary will fail to match complex passwords, rendering the attack ineffective.

B. Password Complexity and Strength

Password complexity is another crucial factor in determining the outcome of a dictionary attack. Passwords that involve a mix of uppercase and lowercase letters, numbers, and symbols are harder to crack compared to simpler ones. As highlighted by Heninger et al. (2010), a password that's long and random will significantly increase the difficulty of cracking attempts. For example, passwords like "LJf45hG!d@2tZ" (a long and random combination) are resistant to dictionary attacks because they fall outside the scope of typical dictionary entries. This emphasizes the importance of using strong passwords in encryption scenarios to ensure robust data protection.

C. Hashcat and Computational Resources

Hashcat is a highly efficient password-cracking tool that benefits from GPU acceleration, allowing it to test a vast number of potential passwords rapidly. According to Kelley et al. (2019), tools like Hashcat provide a significant advantage over traditional CPU-based tools by utilizing parallel processing capabilities of modern GPUs. This allows faster cracking of password hashes, especially when dealing with large dictionaries or more extensive password combinations. However, as with all cracking methods, the efficiency of Hashcat is limited by the quality and diversity of the dictionary. If the correct password isn't included in the dictionary, no amount of computational power can recover it.

D. Time and Computational Effort

The time required to crack a password depends largely on its complexity and the size of the dictionary used. Simple passwords can be cracked in a short amount of time, whereas complex passwords or larger dictionaries can significantly increase the cracking time. As discussed by Kelley et al. (2019), advanced cracking techniques such as mask attacks (where the pattern of characters is known) or rule-based attacks (which modify dictionary entries) can improve the chances of success. However, these techniques require significant computational resources, which may not always be available for every user. The larger the dictionary and the more complex the password, the greater the time and computational effort required.

E. Alternative Attack Methods

When a dictionary attack fails, alternatives like brute-force or hybrid attacks can be attempted. Brute-force attacks test all possible combinations, while hybrid attacks combine dictionary and brute-force methods. Although these methods increase the chances of cracking a password, they are computationally expensive and can take considerable time, especially for long, complex passwords. As Lamarca (2016) mentions, brute-force attacks are only feasible for short and simple passwords, while hybrid methods might succeed when combined with multiple strategies. However, they remain time-consuming and inefficient against highly complex passwords.

VII. CONCLUSION

The experiment successfully demonstrated that free tools like bitlocker2john and Hashcat can be used effectively to crack BitLocker-encrypted drives, provided the passwords are not excessively complex. In comparison, Passware, a commercial alternative, offers a more polished and comprehensive solution, but comes with a significant financial cost, ranging from $3,000 to $5,000 annually. For individuals and smaller organizations, bitlocker2john and Hashcat offer a powerful, cost-free solution to password recovery, providing similar efficiency for many common password scenarios. Passware may be a more convenient option for enterprises due to its support and broader capabilities, but for those on a budget, open-source tools present a viable alternative without compromising on performance.

VIII. ACKNOWLEDGMENT

I would like to express my sincere gratitude to everyone who supported and contributed to the completion of this research paper.
I would also like to thank the John the Ripper Development Team for creating the bitlocker2john tool, which was fundamental for extracting password hashes from BitLocker-encrypted volumes. The tool’s effectiveness in facilitating hash extraction directly contributed to the success of the experiment.
My sincere thanks also go to the Hashcat Community for developing Hashcat, a powerful tool used to crack passwords with GPU acceleration. This tool played a pivotal role in executing the dictionary attack and testing its effectiveness. The team’s dedication to creating such a robust and efficient tool was integral to the success of this research.
Additionally, I am grateful to the authors and researchers whose work laid the foundation for this study. I would like to especially recognize the work of Bonneau (2012), Heninger et al. (2010), and Kelley et al. (2019), whose research on password strength, cracking methods, and security vulnerabilities provided significant insights into the context of this paper.
I am also thankful to my peers and colleagues for their insightful feedback and discussions that helped refine the analysis and interpretation of results. Their suggestions and comments were instrumental in improving the quality of this research.
Lastly, I would like to acknowledge my peers and mentors for their insightful feedback and support throughout the process of conducting this experiment and preparing the report. Their guidance has been invaluable in enhancing my understanding of encryption and password cracking techniques.

REFERENCES

[1] Hashcat. (n.d.). Hashcat - Advanced password recovery. Retrieved from https://hashcat.net/hashcat/.
[2] Openwall. (n.d.). John the Ripper. Retrieved from https://www.openwall.com/john/.
[3] Bonneau, J. (2012). The security of personal data in the cloud. In Proceedings of the 3rd USENIX conference on Hot Topics in Security (HotSec '12). USENIX Association.
[4] Heninger, N., Durbin, M., Mitchell, J. C., & Shacham, H. (2010). Mining your Ps and Qs: Detection of widespread weak keys in network devices. In Proceedings of the 19th USENIX Security Symposium (pp. 233-248). USENIX Association.
[5] Kelley, P., He, Q., & Shacham, H. (2019). Hashcat: The performance of GPU-based password cracking. In Proceedings of the 2019 IEEE European Symposium on Security and Privacy (pp. 273-288). IEEE.
[6] Microsoft. (2020). BitLocker Drive Encryption Overview. Retrieved from https://docs.microsoft.com/en-us/windows/security/information-protection/bitlocker/bitlocker-drive-encryption-overview
[7] S. Kahn, D. (2020). History of encryption and its impact on modern security systems. In Proceedings of the 2020 IEEE Symposium on Security and Privacy (pp. 124-139). IEEE.
[8] Chakraborty, P., & Roy, S. (2020). The evolution of brute-force and dictionary-based password cracking algorithms. Journal of Security and Privacy, 4(4), 175-194.