Experiment No 1

1.1 Aim:
File Hashing for Integrity Verification: Understand the concept of hashing and its use in verifying file integrity.

1.2 Course Outcome:
Apply foundational security principles and cryptographic solutions to protect systems and data.

1.3 Lab Objective:
To understand and demonstrate the concept of cryptographic hashing and how it is used to verify the integrity of files by comparing their hash values.

1.4 Requirements:

● OS: Windows/Linux/macOS

● Hash tools (e.g., md5sum, sha256sum, CertUtil) or Python (hashlib)

● Sample files (.txt, .pdf, etc.)

1.5 Theory:
Hashing is a process of converting data into a fixed-size string of characters, which is typically a sequence of numbers generated from a mathematical algorithm. This string is known as the hash value or digest.

Cryptographic hash functions (such as MD5, SHA-1, SHA-256) are widely used in information security for ensuring data integrity. When a file is processed through a hash function, it produces a hash value. If the file is altered in any way (even by a single bit), the resulting hash value changes significantly. This property makes hash functions ideal for verifying data integrity.

Applications of File Hashing in Integrity Verification:

● Data integrity checks: Verifying that files have not been tampered with during transfer or storage.

● Digital signatures and certificates: Ensuring authenticity.

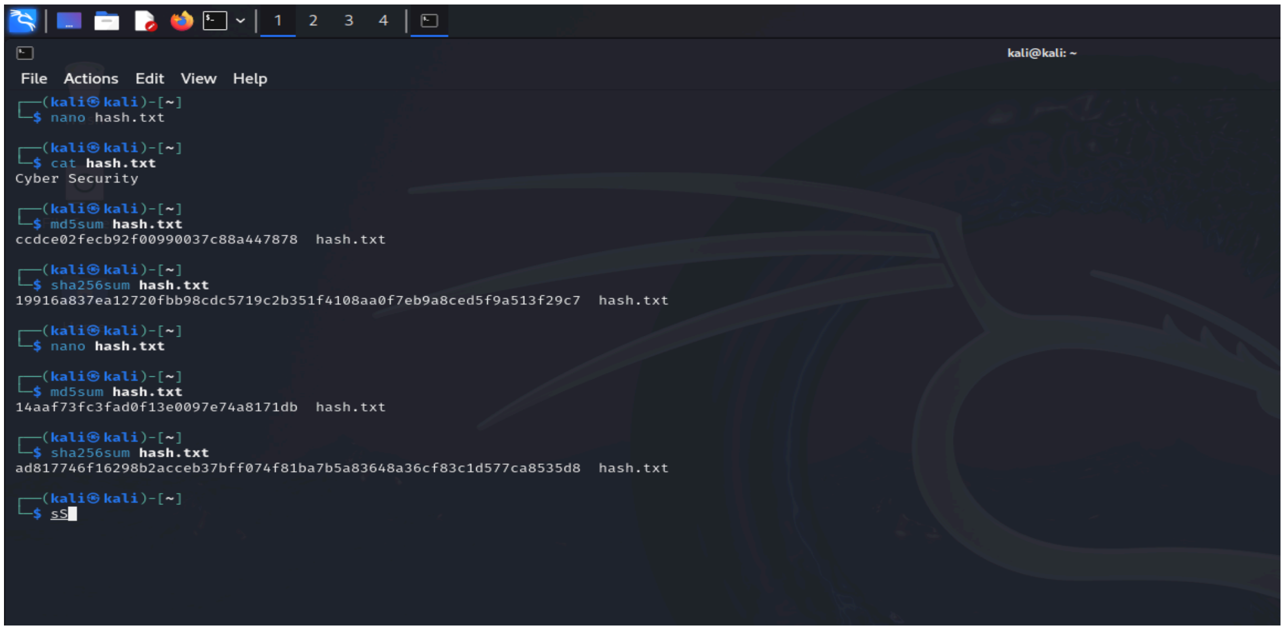
● Password storage: Storing hashed passwords to enhance security.

● Version control systems: Detecting changes in files.

Key Properties of Cryptographic Hash Functions:

1. Deterministic: Same input always yields the same output.
2. Fast computation: Can compute the hash value quickly.
3. Pre-image resistance: Hard to reverse-engineer the input from the hash. 4.
Collision resistance: Hard to find two different inputs that produce the same hash. 5.
Avalanche effect: Small changes in input produce significantly different hashes.

1.6 Program Output:



```
┌──(kali㉿kali)-[~]
└─$ nano hash.txt

┌──(kali㉿kali)-[~]
└─$ cat hash.txt
Cyber Security

┌──(kali㉿kali)-[~]
└─$ md5sum hash.txt
ccdce02fecb92f00990037c88a447878  hash.txt

┌──(kali㉿kali)-[~]
└─$ sha256sum hash.txt
19916a837ea12720fbb98cdc5719c2b351f4108aa0f7eb9a8ced5f9a513f29c7  hash.txt

┌──(kali㉿kali)-[~]
└─$ nano hash.txt

┌──(kali㉿kali)-[~]
└─$ md5sum hash.txt
14aaf73fc3fad0f13e0097e74a8171db  hash.txt

┌──(kali㉿kali)-[~]
└─$ sha256sum hash.txt
ad817746f16298b2acceb37bff074f81ba7b5a83648a36cf83c1d577ca8535d8  hash.txt

┌──(kali㉿kali)-[~]
└─$ sS
```

1.7 Conclusion:

The experiment successfully demonstrated that both MD5 and SHA-256 hashing algorithms produce unique hash values that change significantly when even a small alteration is made to the original text. This confirms the sensitivity and reliability of these hashing algorithms in detecting any modifications. Therefore, hashing serves as an effective method for verifying data integrity and ensuring that files remain unaltered. SHA-256, being more secure, provides stronger resistance against collisions compared to MD5.