

**Ministry of Higher Education and Scientific Research**  
**Northern Technical University**  
**Technical Engineering College of Mosul**  
**Cybersecurity and Cloud Computing Techniques Engineering**

# **Hashing Functions**

**Submitted by:**

**مصطفى عبد العزيز حمزة**

## Introduction

Hash functions, such as MD5, SHA-1, SHA-256, SHA-3, and BLAKE2, play a crucial role in various cryptographic applications. These functions are used for digital signatures, public-key encryption, integrity verification, message authentication, password protection, key agreement protocols, and more.

In the real world, hash functions find widespread use across different domains:

1. Cloud Storage Systems: They help identify identical files and detect modifications.
2. Git Revision Control System: Hash functions are used to uniquely identify files within a repository.
3. Host-Based Intrusion Detection Systems (HIDS): These systems use hash values to detect changes in files.
4. Network-Based Intrusion Detection Systems (NIDS): Hashes are employed to identify known malicious data passing through a network.
5. Forensic Analysis: Hash values serve as evidence that digital artifacts remain unaltered.
6. Bitcoin: The proof-of-work system relies on hash functions.

## How Hash Functions Work

Hash functions take an input of variable length and produce a fixed-size output, commonly referred to as a “digest.” Their primary purpose is to protect data integrity. Unlike ciphers that ensure data confidentiality, hash functions determine whether data has been modified. They achieve this by generating unique digests for each piece of data. Ideally, two distinct pieces of data should never correspond to the same digest. If they do, the hash function is considered vulnerable to collisions.

Additionally, hash functions enhance password security. When storing passwords in a database, they create digests. Even if a data breach occurs, these digests cannot be reverse-engineered to reveal the original input.

## Playing with Python and Hashlib

Python language provides a built-in module called hashlib which contains many hashing algorithms such like: sha256, sha128, and md5.

For example, to generate a sha256 hash for the word 'Plaintext'.

```
[1]: import hashlib  
  
[2]: hashlib.sha256('Plaintext'.encode()).hexdigest()  
  
[2]: '0707c5d972a7029d1696f45c9268cc1dbe2215ae2d6245724f46adc7fd998c46'
```

I programmed a Python tool to carry out hashing and brute-forcing processes, using the following Python modules

- 1- Hashlib for hashing.
- 2- Argparse to parse command-line arguments.
- 3- OS for operating system-related purposes
- 4- Other modules including: pandas, sys, colorama.

For the code from GitHub, scan the QR Code.



## Lab

### 1. Show a help message

```
f:mustafa@kali: ~/Python/Lab/Hashing
(mustafa@kali)-[~/Python/Lab/Hashing]
$ python3 hash.py -h
usage: hash.py [-h] [-p PLAINTEXT] [-a ALGORITHM] [-m] [-b] [-d DICTIONARY] [-c HASH]

This tool can be used to carry out certain hashing functionality against an input plaintext

options:
  -h, --help            show this help message and exit
  -p PLAINTEXT, --PlainText PLAINTEXT
                        The plain text to be hashed.
  -a ALGORITHM, --Algorithm ALGORITHM
                        The hashing algorithm ID to be used.
  -m, --Modules          List all available hashing algorithms along with their IDs.
  -b, --Bruteforcing    To enable bruteforcing mode.
  -d DICTIONARY, --Dictionary DICTIONARY
                        Provide a wordlist to be used with bruteforcing.
  -c HASH, --Hash HASH  Hashed text
```

### 2. To list the available modules

```
(mustafa@kali)-[~/Python/Lab/Hashing]
$ python3 hash.py -m
ID
Algorithm
sha256.      1
sha224.      2
sha3_224.    3
sha384       4
shake_128    5
shake_256    6
sha3_384     7
md5          8
sha1         9
sha3_256     10
blake2b      11
blake2s      12
sha3_512     13
sha512       14
```

### 3. To generate a sha256 hash for the word 'Cybersecurity'

```
(mustafa@kali)-[~/Python/Lab/Hashing]
$ python3 hash.py -p Cybersecurity -a 1
23b2d5dae0a87cf945f37093b119d20fc9931de013bf2836d8e3a2e5699d10ac
```

4. To generate a shake\_128 hash for the words: "Never trust, always verify."

```
(mustafa@kali)-[~/Python/Lab/Hashing]
$ python3 hash.py -p "Never trust, always verify." -a 5
f4a3f8682d285cd37e04f2f440801c377ea69b19b728fa5aeb4c53f9b89b192ee1c1e0d4f7540abc39ab0fc48525dcbaae25bc59736faa610484a38
83451e2eb119842df62beac346301bb1a5ac79ccefcc94e287e4db8f06bc7a001031e043106bd12de567c2743961007cdd634ccb5c36ea55c448c03
e28332e31204fe0f26
```

- 5- To brute force the sha256 hash which was early generated. Using the rockyou wordlist.

```
(mustafa@kali)-[~/Python/Lab/Hashing]
$ echo Cybersecurity >> rockyou.txt

(mustafa@kali)-[~/Python/Lab/Hashing]
$ python3 hash.py -b -c 23b2d5dae0a87cf945f37093b119d20fc9931de013bf2836d8e3a2e5699d10ac -d /home/mustafa/Python/Lab/Hashing/rockyou.txt -a 1
The plaintext Cybersecurity corresponds to the hash [23b2d5dae0a87cf945f37093b119d20fc9931de013bf2836d8e3a2e5699d10ac]
```

- 6- To brute force the blake2s hash:  
**`93f9b0b205276f1c3b03464d5e890d6c99e7e22eb4d7530e53ba2e3a8066d103`**

```
(mustafa@kali)-[~/Python/Lab/Hashing]
$ python3 hash.py -p 12345678 -a 12
93f9b0b205276f1c3b03464d5e890d6c99e7e22eb4d7530e53ba2e3a8066d103

(mustafa@kali)-[~/Python/Lab/Hashing]
$ python3 hash.py -b -c 93f9b0b205276f1c3b03464d5e890d6c99e7e22eb4d7530e53ba2e3a8066d103 -a 12 -d rockyou.txt
The plaintext 12345678 corresponds to the hash [93f9b0b205276f1c3b03464d5e890d6c99e7e22eb4d7530e53ba2e3a8066d103]

(mustafa@kali)-[~/Python/Lab/Hashing]
```