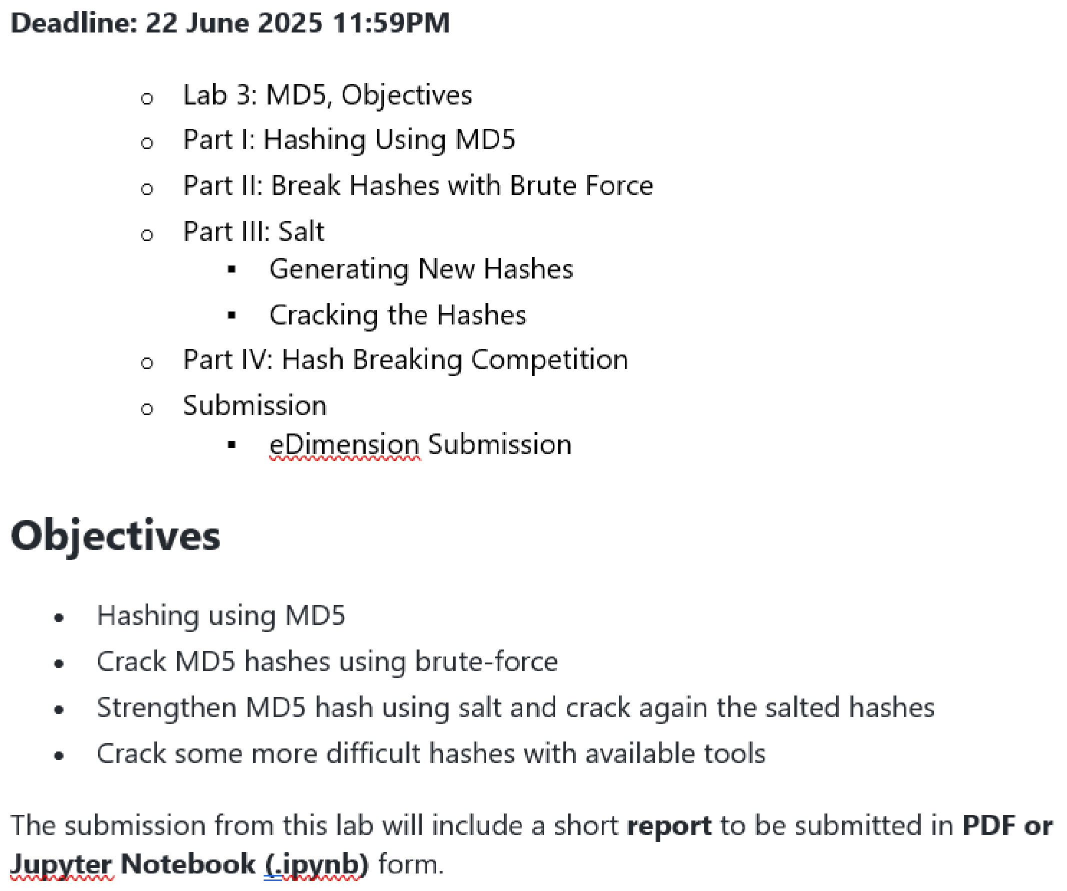
# Lab 3: Hashes

(From **Lab3_2025.pdf)**



**Deadline: 22 June 2025 11:59PM**

- Lab 3: MD5, Objectives
- Part I: Hashing Using MD5
- Part II: Break Hashes with Brute Force
- Part III: Salt
  - Generating New Hashes
  - Cracking the Hashes
- Part IV: Hash Breaking Competition
- Submission
  - eDimension Submission

## Objectives

- Hashing using MD5
- Crack MD5 hashes using brute-force
- Strengthen MD5 hash using salt and crack again the salted hashes
- Crack some more difficult hashes with available tools

The submission from this lab will include a short **report** to be submitted in **PDF or Jupyter Notebook (.ipynb)** form.

## Part I: Hashing Using MD5

In this section, we will explore hashing using MD5. https://en.wikipedia.org/wiki/MD5

Compute a couple of MD5 hashes of a few strings of your choice. The strings should be of different lengths.

There are several methods to generate MD5 hashes. Two are provided here for you to explore:

1. Shell: `echo -n "foobar" | md5sum`
2. Python: `hashlib` library (see example below)

```
import hashlib

plaintext = "Pancakes"

result = hashlib.md5(plaintext.encode())

 print(result.hexdigest())
```

Answer the following questions in your report:

- How does the length of the hash correspond to the input string?
- Are there any visible correlations between the hash and the input string?
- What are the issues related to the cryptographic weakness of MD5?

## Part II: Break Hashes with Brute Force

There are fifteen hash values in the file `hash5.txt` (newline separated).
For this exercise, create a Python 3 script `ex2.py` to reverse the hashes via brute force.
You can achieve this by hashing different input values until you find a hash that matches one in `hash5.txt`.
You need only to consider strings that fulfill the following criteria:

- Length of 5 characters
- Each character **can be a lowercase alphabet or numeric**

Take note of the computation time of your script to reverse all fifteen hashes.

**Save all the hashes you recover as `ex2_hash.txt`**
Answer the following questions in your report:

- How much time did you take in total?
- How much time does it take to crack each string, on average?
- Is it possible to amortize (gradually write off the initial cost of) the brute forcing attempts?

## Part III: Salt

In this section, we deal with salt, but not the kind we are most familiar with.

### Generating New Hashes

You will need to write a script `ex3.py` to do the following:

1. Load the hashes from `ex2_hash.txt` and append one **random** lowercase character as salt value to all the elements of the list of passwords you recovered in the previous part of this exercise.

2. Rehash the password using MD5, and store the newly hashed passwords and their salt values into a new file called `salted6.txt`. Store the new plaintexts as well in a `plain6.txt` file.

The functional definition of our salt strategy is the following:

```
saltedhash(plaintext) = hash(plaintext||salt)
```

Where || denotes concatenation.

Answer the following questions in your report:

- For this exercise, modify the previous Python 3 script `ex2.py` to **ex3.py** to reverse the salted hashes via brute force.
- What are the observed differences between your ease of cracking the salted vs the unsalted plaintexts?
- Report the difference in time taken to crack the salted and the unsalted hash values.
- Explain any differences between salted and un-salted crack strategies.

## Part IV: Hash Breaking Competition

We have provided a list of hashes in `hashes.txt:`

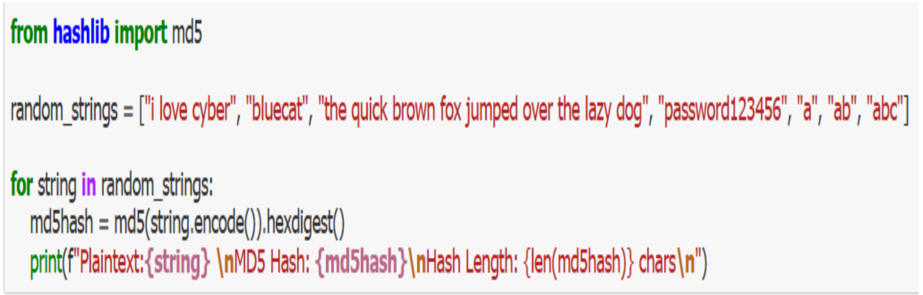
- They are of different difficulty, not all are equally hard.
- There are no easy rules about length or characters allowed any more!
- Try to reverse as many of those hashes as possible. You can also use any other tools as you want. (There are no limitations here)

Submit the answers as `ex4.csv` file containing two columns. The first column is the md5 hash of the plaintext you break, and the second column is the plaintext.
Answer the following questions in your report:

- What is the approach you used to crack the hashes
- How you decided or designed your approach
- Main challenges and limitations of your approach
- **How many hashes out of the total did you manage to crack** (there is no prize)

## Part I: Hashing Using MD5

**Essential file: -**
**Expected result file: -**



```python
from hashlib import md5

random_strings = ["i love cyber", "bluecat", "the quick brown fox jumped over the lazy dog", "password123456", "a", "ab", "abc"]

for string in random_strings:
   md5hash = md5(string.encode()).hexdigest()
   print(f"Plaintext:{string} \nMD5 Hash: {md5hash}\nHash Length: {len(md5hash)} chars\n")
```

## Part II: Break Hashes with Brute Force

**Essential file:** hash5.txt

hash5 Text Document

**Expected result files:** ex2.py + ex2_hash.txt

ex2 Python Source File

ex2_hash Text Document

## Part III: Salt

**Essential file:** ex2_hash.txt

ex2_hash Text Document

**Expected result files:** ex3.py + salted6.txt + plain6.txt

ex3.py Python Source File

salted6 Text Document

plain6 Text Document

## Part IV: Hash Breaking Competition

**Essential file:** hashes.txt

hashes Text Document

**Expected result file:** ex4.csv

ex4.csv Microsoft Excel Comma Separated Values File

## Submission

### eDimension Submission

Lab 3 submission:

Upload a **zip file** with the following: file name format: **lab3_name_studentid .**

**NOTE**: **The file names  must not be changed, and you must name the files as stated in this document.**

**Report (PDF or HTML format)**

- `ex2.py`
- `ex2_hash.txt`
- `ex3.py`
- `salted6.txt`
- `plain6.txt`
- `ex4.csv`
- Jupyter Notebook report (with the outputs saved) in (.ipynb) or (.pdf)

**Deadline: 22 June 2025 11:59PM**