**Lab:** Lab 2 — Classical Cipher Cracking

**Name:** Amit Kumar Sharma
**Registration:** 2020831009

---

# 1. Objective

This lab demonstrates practical attacks against two classic cryptosystems: the **Caesar cipher** and **monoalphabetic substitution ciphers**. The goal is to implement scripts to recover plaintext from ciphertexts, analyze weaknesses, and document the complete thought process.

---

# 2. Environment & Tools

- **Language:** Python 3.8+
- **Editor:** Visual Studio Code (VS Code)
- **Platform:** Windows (PowerShell / Command Prompt examples shown)

---

# 3. Project Structure

```
Lab 2/
 ├── src/
 │    ├── caesar_cracker.py
 │    ├── substitution_solver.py
 │    └── utils.py
 ├── data/
 │    ├── cipher_caesar.txt
 │    ├── cipher_sub_1.txt
 │    └── cipher_sub_2.txt
 ├── wordlists/
 │    └── common_words.txt
 ├── outputs/
 │    ├── caesar_plain.txt
 │    ├── sub1_attempt.txt
 │     └── sub2_attempt.txt
 ├── REPORT.md (this file)
 └── README.md
```

---

## 4. How to run (Commands)

Open a terminal and change to the project folder. Example (replace path if needed):

```
cd "E:\INS_Lab_Task\Lab 2"
```

### Run the Caesar cracker:

```
python "src/caesar_cracker.py" "data/cipher_caesar.txt"
"outputs/caesar_plain.txt"
```

### Run the substitution solver for Cipher 1:

```
python "src/substitution_solver.py" "data/cipher_sub_1.txt"
"outputs/sub1_attempt.txt"
```

### Run the substitution solver for Cipher 2:

```
python "src/substitution_solver.py" "data/cipher_sub_2.txt"
"outputs/sub2_attempt.txt"
```

### View outputs in terminal (Windows):

```
type "outputs\caesar_plain.txt"
type "outputs\sub1_attempt.txt"
type "outputs\sub2_attempt.txt"
```

---

## 5. Implementation Notes

### caesar_cracker.py

- Performs a full key-space search (shift 0–25).
- Prints all candidate plaintexts and optionally scores them by English word frequency.

- Writes the most-likely plaintext to the specified output file.

### substitution_solver.py

- Uses a hill-climbing / simulated annealing style approach:
    - Start with an initial guess mapping (e.g., frequency-based).
    - Apply small random swaps to the key and accept improvements by score.
    - Score candidate plaintexts using an English language model (wordlist matches and n-gram scoring).
- Writes the best-found plaintext and the mapping to the output file.

### utils.py

- Utility functions: file I/O, scoring functions, wordlist loader, n-gram frequency tables.

---

## 6. Example Outputs (summary)

- `outputs/caesar_plain.txt` — recovered plaintext for Caesar-cipher file.
- `outputs/sub1_attempt.txt` — best plaintext attempt for substitution cipher 1.
- `outputs/sub2_attempt.txt` — best plaintext attempt for substitution cipher 2.

Include these outputs as appendices or attachments in your submission.

---

## 7. Observations & Analysis

1. **Caesar Cipher**
    - Vulnerable to brute-force due to only 26 keys.
    - Automatic scoring using common words quickly identifies the correct shift.
2. **Monoalphabetic Substitution**

    - Much harder than Caesar; key space is 26! (impractical to brute-force).
    - Heuristic/search methods (hill-climbing with n-gram scoring) work well for medium-length ciphertexts.
    - Frequency analysis provides a good starting point, but local search is needed to refine the mapping.

3. **Common Failure Modes**
    - Very short ciphertexts produce poor frequency statistics → solvers may fail or return many plausible candidates.
    - Proper scoring (wordlist matches + quadgram scoring) improves reliability.