Lab 2 Report: Attacking Classic Crypto Systems

Checkpoint 1: Caesar Cipher Attack

Ciphertext

```
odroboewscdrolocdcwkbdmyxdbkmdzvkdpybwyeddrobo
```

Summary

This ciphertext was produced with the Caesar cipher, a simple substitution method that shifts each plaintext letter by a fixed number of alphabet positions. Because the number of possible shifts is small, the cipher is vulnerable to exhaustive search.

Method

Attack strategy — brute force.
The Caesar cipher has only 26 possible keys (shifts 0–25). The attack exhaustively applies every shift to the ciphertext and inspects the outputs to locate a meaningful English message.

Decryption procedure.

- Iterate over candidate shifts from 0 to 25.
- For each candidate: for every character in the ciphertext, if it is an alphabetic character shift it backwards by the candidate amount, wrapping around the alphabet (use modulo 26). Leave non-letters unchanged.
- Collect each candidate plaintext for evaluation.

Selecting the correct key.
Evaluate the 26 candidate plaintexts and identify the one that forms readable English by looking for common words, natural phrase structure, and overall coherence.

Implementation (brief)

A compact Python routine was used to:

1. Loop through all 26 shifts.
2. Apply a left-shift decryption to letters with proper wrap-around.
3. Print the resulting plaintext candidates for manual review.

Result

The correct plaintext was produced using a shift of **10**.

Decrypted output:
`ethereumisthebestsmartcontractplatformoutthere`

In plain words: “ethereum is the best smart contract platform out there.”

Discussion

The ease of recovering the plaintext demonstrates the Caesar cipher’s practical weaknesses:

- **Tiny keyspace.** With only 26 possible keys, brute force is trivial.
- **No key secrecy or complexity.** The key is simply a small integer.
- **Static mapping.** Each letter maps consistently to a single ciphertext letter.
- **Preserved structure.** Word patterns and letter frequencies are retained, aiding recognition.

Checkpoint 2: Substitution Cipher Report

Cipher 1 & Cipher 2

Objective

The goal of this checkpoint is to decrypt two monoalphabetic substitution ciphers (`cipher1.txt` and `cipher2.txt`) using a Python-based hill-climbing approach guided by English letter frequency scoring.

Approach Summary

1. **Frequency Analysis**
    - Initial guesses for substitution keys were generated randomly.
    - English letter frequencies were used to score candidate plaintexts.
    - Common English letters like `E`, `T`, `A`, `O`, `I`, `N`, and `S` were used to guide scoring.
2. **Hill-Climbing Algorithm**

- A random key (substitution of 26 letters) was generated.
- Iteratively, two letters were swapped to create a new candidate key.
- The plaintext produced by the candidate key was scored using letter frequency similarity to English.
- If the candidate score was better than the previous best, the key was accepted.
- Simulated annealing was applied by allowing some worse keys to be accepted probabilistically, helping avoid local maxima.

3. **Plaintext Scoring**
    - For each candidate decryption:
        - Count occurrences of all letters.
        - Compare frequency percentages with standard English frequencies.
        - Compute a score: lower absolute differences yield higher scores.
    - The key producing the highest score was chosen as the final decryption.
4. **Output**
    - The program outputs:
        - Best candidate plaintext
        - Final key mapping (cipher → plain letter)
    - Iteration limits were set to 6000–8000 to allow sufficient exploration of the key space.

Cipher 1 Solver

```
python3 checkpoint2_cipher1.py cipher1.txt > out1.txt
```

- Key features:
    - Hill-climbing with simulated annealing.
    - Temperature decay: `temperature *= 0.999`.
    - 6000 iterations for convergence.

    - English letter frequency scoring.

Cipher 2 Solver

```
python3 checkpoint2_cipher2.py cipher2.txt > out2.txt
```

- Key features:
    - Similar hill-climbing with annealing.
    - Slightly higher iterations: 8000 for better exploration.
    - Temperature starts at 7.0 to allow broader swaps initially.
    - Outputs plaintext and key mapping.

Steps Taken

1. Loaded the ciphertext from the file and converted to uppercase.
2. Removed all non-alphabet characters to simplify scoring.
3. Generated a random substitution key.
4. Applied hill-climbing with the following loop:
    - Mutate the key by swapping two letters.
    - Apply the key to decrypt the text.
    - Score the plaintext against English letter frequencies.
    - Accept the new key if score improves or probabilistically (simulated annealing).
5. Repeated the iteration to refine the key.
6. Returned the best scoring plaintext and the corresponding key.

Results

Cipher 1

- Output file: `out1.txt`
- Candidate plaintext
- Key mapping:

```
A -> X, B -> T, C -> E, ...
```

Cipher 2

- Output file: `out2.txt`
- Candidate plaintext
- Key mapping:

```
A -> M, B -> A, C -> F, ...
```

Analysis

- **Easier Cipher**: Cipher 2 was easier to break.
    - Reason: Slightly longer and contained more repetitive patterns and common words.
    - Higher convergence score observed in the hill-climbing iterations.
- **Cipher 1**:
    - Shorter ciphertext with fewer repeating patterns.
    - Required more iterations to reach clarity.
- **Observations**:
    - Hill-climbing converged faster when common letters (`E`, `T`, `A`) were placed correctly.
    - Simulated annealing helped escape local maxima, improving final plaintext quality.

Conclusion

- Both ciphers were successfully decrypted using **hill-climbing substitution solver**.
- English letter frequency scoring proved effective for this task.

- The Python scripts provide:
    - Automatic key search
    - Plaintext candidate
    - Key mapping
- This approach can be further improved using:
    - Quadgram or dictionary-based scoring
    - Multiple restarts for higher confidence in the best key