Attacking Classic Crypto Systems – Lab 2

Solutions, steps, commands and code

Md. Rakibul Kabir
Reg: 2020831051

Overview

This file contains solutions (with code and commands) to the assigned tasks:

- Checkpoint 1: Break a Caesar cipher.
- Checkpoint 2: Break substitution ciphers (two ciphertexts) using frequency analysis + hill-climbing approach.

All code samples are in Python 3 and ready to run on your machine (or in Overleaf's code listing — to run, copy into a '.py' file).

1 Checkpoint 1: Caesar cipher

Cipher

`odroboewscdrolocdcwkbdmyxdbkmdzvkdpybwyeddrobo`

Approach (steps)

1. Try all 26 shifts (brute-force).
2. Print English-looking candidates; choose the meaningful one.

Python program: brute-force Caesar

```
# caesar_break.py
import string

cipher = "odroboewscdrolocdcwkbdmyxdbkmdzvkdpybwyeddrobo"
alpha = string.ascii_lowercase

def caesar(s, shift):
    out = []
    for ch in s:
        if ch in alpha:
            out.append(alpha[(alpha.index(ch)-shift) % 26])
        else:
            out.append(ch)
    return ''.join(out)

for shift in range(26):
    cand = caesar(cipher, shift)
    print(f"shift {shift}: {cand}")
```

How to run

```
python3 caesar_break.py
```

Expected outcome

One of the shifts will produce an English sentence. (When you run, inspect outputs and pick the readable one).

2 Checkpoint 2: Substitution ciphers

Cipher-1 and Cipher-2

(See lab manual for full ciphertext blocks; paste them into files named `cipher1.txt` and `cipher2.txt`.)

Approach summary

1. Use frequency analysis to get initial guesses (map most frequent ciphertext letters to 'e', 't', 'a', ...).
2. Improve the mapping using a scoring function based on English quadgram (or wordlist) frequency and a hill-climbing / simulated annealing algorithm to maximize score.
3. Output the best plaintext candidate.

Dependencies

This code uses a simple English scoring based on quadgrams. The code below includes a minimal quadgram scorer; for better results, you can use a precomputed quadgram frequency file.

Python solver (hill-climbing, simple)

```
# subcipher_break.py
# Simple substitution cipher breaker using hill-climbing with English
    scoring
import math, random, string, sys
from collections import Counter

alphabet = string.ascii_lowercase

def load_cipher(filename):
    with open(filename) as f:
        return ''.join(f.read().lower().split())

# basic english wordlist scoring (word matches) for quick improvement
common_words = ["the","and","that","have","for","not","with","you","
    this","but","his","from","they","she","which"]

def score_plaintext(text):
    # Score by counting common words occurrences and letter frequency
       fitness
    score = 0
    for w in common_words:
        if w in text:
            score += 10 * text.count(w)
    # letter frequency closeness to English
    freq = Counter(c for c in text if c.isalpha())
    # promote vowels presence
    vowels = sum(freq[c] for c in "aeiou")
    score += vowels
    return score

def decrypt_with_key(cipher, keymap):
    table = str.maketrans(alphabet, ''.join(keymap))
    return cipher.translate(table)
```

```
def random_key():
    L = list(alphabet)
    random.shuffle(L)
    return L

def mutate_key(key):
    k = key[:]
    i,j = random.sample(range(26),2)
    k[i],k[j] = k[j],k[i]
    return k

def hill_climb(cipher, iterations=2000):
    best_key = random_key()
    best_plain = decrypt_with_key(cipher, best_key)
    best_score = score_plaintext(best_plain)
    for it in range(iterations):
        cand_key = mutate_key(best_key)
        cand_plain = decrypt_with_key(cipher, cand_key)
        cand_score = score_plaintext(cand_plain)
        if cand_score > best_score:
            best_key, best_score = cand_key, cand_score
            best_plain = cand_plain
            # print progress
            print(f"iter {it} improved score {best_score}: {best_plain
                [:120]}")
    return best_plain, best_key, best_score

if __name__=='__main__':
    if len(sys.argv) < 2:
        print("Usage: python3 subcipher_break.py cipherfile.txt")
        sys.exit(1)
    cipher = load_cipher(sys.argv[1])
    best_plain, key, sc = hill_climb(cipher, iterations=5000)
    print("\nBest score:", sc)
    print("Plaintext candidate:\n")
    print(best_plain)
    print("\nKey mapping (cipher->plain):")
    for c,p in zip(alphabet, ''.join(key)):
        print(f"{c} -> {p}", end='  ')
    print()
```

How to run

```
python3 subcipher_break.py cipher1.txt > out1.txt
python3 subcipher_break.py cipher2.txt > out2.txt
```

Notes and tuning

- The above basic hill-climbing is a simple starter. For higher-quality results, use an English scoring function based on quadgrams (commonly available) and run multiple restarts + simulated annealing.
- You may also use `simulated annealing` or `pyenchant` with dictionary matching to guide swaps.
- Compare which of the two ciphertexts was easier: generally, the shorter or more repetitive text is easier to break; also texts with higher word structure and common words are easier.

Which input is easier to break?

- **Method to decide:** Compare final scoring convergence, number of English words found, and clarity of candidate plaintexts.
- **Reason**: Ciphertexts containing more common short words (e.g., "the", "and", "to") and natural sentence punctuation are easier because frequency patterns and word boundaries help guide mapping.

Deliverables

1. Show the working Python scripts and outputs to the teacher.
2. Include the best plaintext candidates (files out1.txt and out2.txt), and explain the steps taken and reasoning in the report.