# CSE-478 Lab 2: Attacking Classic Crypto Systems — Report

---

## Objectives

- Attack two classic cryptosystems (Caesar and monoalphabetic substitution) programmatically.
- Demonstrate weaknesses by recovering plaintexts from provided ciphers.
- Provide checkpoints and a detailed report that explains the approach and thought process.

## Submission Items

- **Checkpoint-1:** Program + result for the Caesar cipher ciphertext.
- **Checkpoint-2:** Programs + results for two monoalphabetic substitution ciphertexts; identify which was easier to break and explain why.
- **This report:** Detailed approach and reasoning for each system.

---

## Checkpoint-1 (Marks 5) — Caesar Cipher

### Provided Cipher

odroboewscdrolocdcwkbdmyxdbkmdzvkdpybwyeddrobo

## Approach

1. **Brute force all 26 shifts** (rotate letters backward). Non-letters are preserved.
2. **Rank/inspect candidates**. We printed all candidates; an n-gram score can auto-select the best, but visual inspection sufficed here.
3. **Select the intelligible English plaintext.**

## Key Code Sketch (from `lab2_checkpoint1.ipynb`)

- `decrypt(cipher, shift)` rotates A–Z/a–z by `shift` using modular arithmetic; keeps punctuation unchanged.
- Loop `shift ∈ [0,25]`, print each candidate; optionally keep the best by dictionary word hits.

## Output (abridged)

Trying all possible shifts:
Shift 0: odroboewscdrolocdcwkbdmyxdbkmdzvkdpybwyeddrobo
Shift 1: ncqnandvrbcqnknbcbvjaclxwcajlcyujcoxavxdccqnan
...
Shift 9: fuifsfvnjtuifcftutnbsudpousbduqmbugpsnpvuuifsf
Shift 10: ethereumisthebestsmartcontractplatformoutthere
Shift 11: dsgdqdtlhrsgdadrsrlzqsbnmsqzbsokzsenqlntssgdqd
...

## Final Answer

- **Shift:** 10
- **Plaintext:** `ethereumisthebestsmartcontractplatformoutthere`

**Thought process:** The candidate at shift 10 forms a fluent English sentence with recognizable words (“ethereum”, “smart contract”, “platform”), which is overwhelmingly more plausible than other shifts.

---

# Checkpoint-2 (Marks 8 + 7) — Substitution Ciphers (Two Inputs)

## General Strategy

We attacked both monoalphabetic substitution ciphers using a **frequency-analysis-first** pipeline augmented by **pattern constraints** and **iterative refinement**:

1. **Frequency analysis:** Count letter frequencies in the ciphertext; map highest-frequency letters toward English frequency order (E, T, A, O, I, N, S, H, R, …).

2. **Word-shape constraints:** Use patterns like one-letter words → {A, I}; common bigrams/digrams (TH, HE, IN, ER); trigrams (THE, ING, AND); punctuation/apostrophe cues.

3. **Dictionary-guided refinement:** After initial mapping, repeatedly substitute and check for emerging words; lock in confident mappings and continue.

4. **Heuristic search (optional):** Where ambiguity remained, consider hill-climbing swaps over the key to improve a quadgram score; simple manual tweaks sufficed for these inputs.

> Implementation note: both `substitution_cipher1.ipynb` and `substitution_cipher_2.ipynb` define a `break_substitution_cipher(cipher: str) -> str` routine that applies the above steps and prints the best-current plaintext.

---

## Cipher-1

**Provided Cipher (excerpt)**

af p xpkcaqvnpk pfg, af ipqe qpri, gauuikifc tpw, ceiri udvk tiki afgarxifrphni cd eao- ...

**Result (from notebook)**

Decryption for Cipher 1:
in a particular and, in each case, different way, these four were indispensable to him- ...

(Excerpt shown; full plaintext recovered in the notebook output.)

**Reasoning Highlights**

- High frequency letters aligned well with **E/T/A/O** ordering.
- Early recognition of common words (“in”, “and”, “the”, “each”, “case”, “different”) quickly stabilized many mappings.
- Cohesive literary prose emerged after a few iterations, confirming correctness.

---

## Cipher-2

**Provided Cipher (excerpt)**
aceah toz puvg vcdl omj puvg yudqecov, omj loj auum klu thmjuv hs klu zlcvu shv zcbkg guovz, ...

**Result (from notebook)**
Decryption for Cipher 2:
bilbo was very rich and very peculiar, and had been the wonder of the shire for sixty years, ...

(Excerpt shown; full plaintext recovered in the notebook output.)

**Reasoning Highlights**

- Distinctive names and phrases (e.g., **“bilbo”, “shire”, “bag end”, “mr. baggins”**) provided strong anchors for letter mapping.
- Proper nouns created quick certainty for multiple letters simultaneously, accelerating convergence.

---

## Which Input Was Easier? Why?

- **Easier: Cipher-2** was easier and faster to break.
- **Explanation:** Presence of **proper nouns** and **highly distinctive collocations** (“bilbo”, “shire”, “bag end”, “mr. baggins”, “well-preserved”) dramatically reduces ambiguity in monoalphabetic substitution. These anchors fix multiple mappings at once. By contrast, **Cipher-1** (expository prose) relied more on generic function words and required more

iterative refinement before fluent text emerged.

---

# Discussion: Weaknesses Demonstrated

- **Caesar cipher** has only 25 effective keys—**trivial to brute-force**.
- **Simple substitution** preserves plaintext **letter frequencies** and **word shapes**; these leak enough information for recovery via frequency analysis + pattern constraints. A small amount of context (proper nouns) makes attacks especially quick.

### Possible Enhancements (if revisiting the code)

- Add **quadgram scoring** to auto-select the best decryption without manual inspection.
- Implement **hill-climbing / simulated annealing** over key permutations for tougher texts.
- Include **unit tests** and a CLI wrapper to reproduce results from raw ciphertexts.

---

# Appendix

- **Functions implemented:**
  - `lab2_checkpoint1.ipynb`: `decrypt(cipher, shift)`, `main`
  - `substitution_cipher1.ipynb`: `break_substitution_cipher(cipher)`
  - `substitution_cipher_2.ipynb`: `break_substitution_cipher(cipher)`
- **Recovered plaintext excerpts shown above;** complete outputs are preserved in the notebooks' output cells.
- **Environment:** Python (Jupyter).