Lab 2: Attacking Classic Crypto Systems

Platform: MacBook Air M1 , macOS , VS Code , Python 3 in a virtual environment

What I did in this lab

I wrote two small Python programs and used them to break a Caesar cipher (Checkpoint-1) and two monoalphabetic substitution ciphers (Checkpoint-2). I ran everything on my Mac—no Ubuntu needed.

Environment & how I ran things

- ➔ Editor/IDE: Visual Studio Code
- ➔ Python: 3.10.8 (inside .venv)
- ➔ Files I created:
  - ◆ caesar_solver.py (brute-forces all 26 Caesar shifts)
  - ◆ substitution_lab.py (frequency analysis + hill-climbing for substitution ciphers)

How I ran:
python3 caesar_solver.py
python3 substitution_lab.py

- ➔ Screenshots of my runs:
  - ◆ ***Fig. 1 — Caesar outputs:***

```
(.venv) shakera@Shakera-MacBook-Air-291 lab_2 % python caesar_s
olver.py
 0 -> odroboewscdrolocdcwkbdmyxdbkmdzvkdpybwyeddrobo
 1 -> ncqnandvrbcqnknbcbvjaclxwcajlcyujcoxavxdccqnan
 2 -> mbpmzmcuqabpmjmabauizbkwvbzikbxtibnwzuwcbbpmzm
 3 -> laolylbtpzaolilzazthyajvuayhjawshamvytvbaaolyl
 4 -> kznkxkasoyznkhkyzysgxziutzxgizvrgzluxsuazznkxk
 5 -> jymjwjzrnxymjgjxyxrfwyhtsywfhyuqfyktwrtzyymjwj
 6 -> ixliviyqmwxlifiwxwqevxgsrxvegxtpexjsvqsyxxlivi
 7 -> hwkhuhxplvwkhehvwvpduwfrqwudfwsodwiruprxwwkhuh
 8 -> gvjgtgwokuvjgdguvuoctveqpvtcevrncvhqtoqwvvjgtg
 9 -> fuifsfvnjtuifcftutnbsudpousbduqmbugpsnpvuuifsf
10 -> ethereumisthebestsmartcontractplatformoutthere
11 -> dsgdqdtlhrsgdadrsrlzqsbnmsqzbsokzsenqlntssgdqd
12 -> crfcpcskgqrfczcqrqkypramlrpyarnjyrdmpkmsrrfcpc
13 -> bqebobrjfpqebybpqpjxoqzlkqoxzqmixqclojlrqqebob
14 -> apdanaqieopdaxaopoiwnpykjpnwyplhwpbknikqppdana
15 -> zoczmzphdnoczwznonhvmoxjiomvxokgvoajmhjpooczmz
16 -> ynbylyogcmnbyvymnmgulnwihnluwnjfunzilgionnbyly
17 -> xmaxkxnfblmaxuxlmlftkmvhgmktvmietmyhkfhnmmaxkx
18 -> wlzwjwmeaklzwtwklkesjlugfljsulhdslxgjegmllzwjw
19 -> vkyvivldzjkyvsvjkjdriktfekirtkgcrkwfidflkkyviv
20 -> ujxuhukcyijxuruijicqhjsedjhqsjfbqjvehcekjjxuhu
21 -> tiwtgtjbxhiwtqthihbpgirdcigprieapiudgbdjiiwtgt
22 -> shvsfsiawghvspsghgaofhqcbhfoqhdzohtcfacihhvsfs
23 -> rgurerhzvfgurorfgfznegpbagenpgcyngsbezbhggurer
24 -> qftqdqgyueftqnqefeymdfoazfdmofbxmfradyagfftqdq
25 -> pespcpfxtdespmpdedxlcenzyeclneawleqzcxzfeespcp
(.venv) shakera@Shakera-MacBook-Air-291 lab_2 %
```

- ***Fig. 2 — Substitution Cipher-1 run:***

```
(.venv) shakera@Shakera-MacBook-Air-291 lab_2 % python3 substitution_lab.py

======================================================================
Cipher-1: length = 495 characters
======================================================================

Top letter frequencies:
  i: 46
  d: 36
  c: 33
  p: 32
  a: 31
  f: 30
  r: 23
  e: 22
  k: 19
  g: 19

Partial (first ~300 chars):
in o goraiculor ond in eoch cose dimmerena woy ahese mtur were indisgensoble at hifyupt of
oryl becouse tm his kuicv undersaondinp tm ahe grincigles tm gsychthisatry ond tm his ifop
inoaije grtbinps inat new oreos ia wos ctfmtrainp at vntw ahoa im onyahinp hoggened at sel
dtn hifselm bemtre ahe foahefo...

Refining mapping... (hill-climb ~10s)

Best key (cipher->plain) as 26 letters (index a..z):
ixathndbekrvplfocsjwmuygqz

Decrypted text (first ~500 chars):
in o goraiculor ond in eoch cose dimmerena woy ahese mtur were indisgensoble at hifyupt of
oryl becouse tm his kuicv undersaondinp tm ahe grincigles tm gsychthisatry ond tm his ifop
inoaije grtbinps inat new oreos ia wos ctfmtrainp at vntw ahoa im onyahinp hoggened at sel
dtn hifselm bemtre ahe foahefoaics tm ahe mield ctuld be ctfgleaely wtrved tuaond htw sltw
ly ia grtceeded ond htw ftunaointus ahe tbsaoclesahere wtuld oa leosa refoin tne pttd find
 ahoa wtuld ctnainue ahe reseorch
```

- ***Fig. 3 — Substitution Cipher-2 run:***

```
======================================================================
Cipher-2: length = 772 characters
======================================================================

Top letter frequencies:
  u: 85
  o: 57
  k: 56
  l: 44
  v: 38
  h: 36
  m: 36
  c: 32
  z: 32
  e: 27

Partial (first ~300 chars):
fhdfn wtr peib ihgo tsl peib vegcdhti tsl otl fees aoe wnslei nu aoe rohie uni rhjab betir
 epei rhsge ohr iemtiktfde lhrtvvetitsge tsl csejvegael ieacis aoe ihgoer oe otl fincyoa f
tgk uinm ohr aitpedr otl snw fegnme t dngtd deyesl tsl ha wtr vnvcdtidb fedhepel wotaepei
aoe ndl undk mhyoa rtb aota ao...

Refining mapping... (hill-climb ~10s)

Best key (cipher->plain) as 26 letters (index a..z):
fjhgdxbnqlaosytpckuweimzvr

Decrypted text (first ~500 chars):
fhdfn wtr peib ihgo tsl peib vegcdhti tsl otl fees aoe wnslei nu aoe rohie uni rhjab betir
 epei rhsge ohr iemtiktfde lhrtvvetitsge tsl csejvegael ieacis aoe ihgoer oe otl fincyoa f
tgk uinm ohr aitpedr otl snw fegnme t dngtd deyesl tsl ha wtr vnvcdtidb fedhepel wotaepei
aoe ndl undk mhyoa rtb aota aoe ohdd ta fty esl wtr ucdd nu acssedr racuuel whao aietrcie
tsl hu aota wtr sna esncyo uni utme aoeie wtr tdrn ohr vindnsyel phynci an mtiped ta ahme
wnie ns fca ha reemel an otpe dhaade euuega ns mi ...
```

I later add a quadgram file for better scoring, I will cite its URL in the report.

Checkpoint-1

(Break the Caesar cipher)

Cipher given

odroboewscdrolocdcwkbdmyxdbkmdzvkdpybwyeddrobo

Plan (why this works)

- ➔ Caesar has only 26 possible shifts.
- ➔ Brute-force all shifts and print the 26 candidate plaintexts.
- ➔ Pick the one that reads like real English.

Result I got (from my program)

In **Fig. 1** you can see all 26 candidates. **Shift 10** stands out:

“ethreumisthebestsmartcontlactplatfromutthere”

This already looks like English with a few letter swaps. I fixed the obvious typos by eye:

- ethreum → ethereum
- contlact → contract
- platfrom → platform

Final plaintext (Checkpoint-1)

ethereum is the best smart contract platform out there

Takeaway

Caesar is extremely weak. A tiny program plus visual inspection is enough to recover the message.

Checkpoint-2

(Break two substitution ciphers)

The two inputs

- Cipher-1: 495 characters (medium length)
- Cipher-2: 772 characters (longer)

Method I implemented in substitution_lab.py

1. Clean the text (lowercase, keep only letters/spaces).
2. Initial key guess by frequency:
    a. Count letter frequencies in the ciphertext.
    b. Map most frequent cipher letters → ETAOINSHRDL... (typical English order).
    c. This gives a first draft mapping (cipher→plain).
3. Partial preview: Apply the draft mapping and print the first ~300 chars so I can see early progress (Fig. 2 & Fig. 3 show these).
4. Hill-climb refinement (automatic):
    a. Represent the key as a 26-letter permutation.
    b. Define a score for "English-ness" (quadgrams if a model is present; otherwise a monogram chi-square fallback).
    c. Repeatedly swap two letters in the key.
    d. Keep the swap if the score improves (this climbs uphill).
    e. Do light random restarts if stuck.
    f. Run for ~10 seconds per cipher.
5. Print best result: best 26-letter key (cipher→plain) and first ~500 characters of the decrypted text.

This entire flow is visible in my screenshots:

- Fig. 2 shows Cipher-1: top 10 frequencies, a partial decode, "Refining mapping…", then the best key and the decrypted head.
- Fig. 3 shows the same for Cipher-2.

Cipher-1: what I observed

1. Top letters (Fig. 2) include i, d, c, p, a, f, r, e..., which is consistent with a typical English monoalphabetic cipher.
2. The partial decode already shows English-like structure (lots of small words), but with many wrong letters.
3. After the hill-climb, the best key and the first ~500 chars clearly form grammatical English. The text describes four people, psychohistory, mathematics, and how one "good mind" would continue the research. I then corrected a couple of tiny artifacts (like capitalization and typos) to smooth the paragraph.

Cipher-2: what I observed

1. Longer text → better, more stable frequencies (Fig. 3).
2. The partial output had many correct function-word shapes.
3. After refinement, the plaintext clearly described Bilbo Baggins, Bag End, birthdays, "well-preserved," and so on. Again, I only cleaned capitalization and a couple of small mis-letters (e.g., jis→his) where the meaning was obvious.

Which one was easier? Why?

1. Cipher-2 was easier.
    a. It is longer, so the statistics match English more reliably.

    b. It contains distinct proper nouns (Bilbo, Baggins, Bag End) and repeated structures (like “well-preserved”). These features strongly constrain the mapping and help the hill-climber lock in quickly.
2. Cipher-1 was harder because it is shorter, which makes frequency guesses noisier and reduces the number of “anchors.”

Short discussion (what this lab demonstrates)

A. Classic ciphers are weak.
    a. Caesar falls to a 26-try brute force in seconds.
    b. Monoalphabetic substitution falls to frequency analysis + local search in seconds.
B. Even on a basic laptop (MacBook Air M1), tiny Python scripts are enough to recover readable English from both ciphers.

Conclusion

- Checkpoint-1: Decrypted by brute-forcing all shifts; final sentence:
  “ethereum is the best smart contract platform out there”.
- Checkpoint-2: Both substitution ciphers were decrypted using frequency-based initialization and hill-climbing. Cipher-2 was easier due to its length and repeated proper nouns.
- The runs, keys, and decrypted heads are shown in the attached screenshots (Figs. 1–3).