Report on Substitution Cipher Decryption

1. Introduction

In this coursework, I was tasked with deciphering a ciphertext that had been encrypted using a substitution cipher. The goal was threefold: (a) determine the encryption key (i.e., the mapping from plaintext letters to ciphertext letters), (b) explain how this key was discovered, and (c) discuss how my approach compares with the method used in Lab 1.

Interesting Observation:

An unexpected discovery was that the decrypted text was a historical account of **Mary, Queen of Scots' trial**, a conspiracy case against **Queen Elizabeth I** that led to Mary's **execution in 1587**. This result not only made the decryption compelling but also demonstrated how historical texts can be **encoded and later reconstructed** using cryptographic techniques.

```
--- Decrypted Plaintext ---
onthemorningofsaturdayoctoberqueenmaryenteredthecrowdedcourtroomatfotheringhaycastleyearsofimprisonm
entandtheonsetofrheumatismhadtakentheirtollyetsheremaineddignifiedcomposedandindisputablyregalassist
edbyherphysicianshemadeherwaypastthejudgesofficialsandspectatorsandapproachedthethronethatstoodhalfw
ayalongthelongnarrowchambermaryhadassumedthatthethronewasagestureofrespecttowardherbutshewasmistaken
thethronesymbolizedtheabsentqueenelizabethmarysenemyandprosecutormarywasgentlyguidedawayfromthethron
eandtowardtheoppositesideoftheroomtothedefendantsseatacrimsonvelvetchairmaryqueenofscotswasontrialfo
rtreasonshehadbeenaccusedofplottingtoassassinatequeenelizabethinordertotaketheenglishcrownforherself
sirfranciswalsinghamelizabethsprincipalsecretaryhadalreadyarrestedtheotherconspiratorsextractedconfe
ssionsandexecutedthemnowheplannedtoprovethatmarywasattheheartoftheplotandwasthereforeequallytoblamea
ndequallydeservingofdeathwalsinghamknewthatbeforehecouldhavemaryexecutedhewouldhavetoconvincequeenel
izabethofherguiltalthoughelizabethdespisedmaryshehadseveralreasonsforbeingreluctanttoseeherputtodeat
hfirstmarywasascottishqueenandmanyquestionedwhetheranenglishcourthadtheauthoritytoexecuteaforeignhea
dofstatesecondexecutingmarymightestablishanawkwardprecedentifthestateisallowedtokillonequeenthenperh
apsrebelsmighthavefewerreservationsaboutkillinganothernamelyelizabeththirdelizabethandmarywerecousin
sandtheirbloodtiemadeelizabethallthemoresqueamishaboutorderingtheexecutioninshortelizabethwouldsanct
ionmarysexecutiononlyifwalsinghamcouldprovebeyondanyhintofdoubtthatshehadbeenpartoftheassassinationp
lottheconspiratorswereagroupofyoungenglishcatholicnoblemenintentonremovingelizabethaprotestantandrep
lacingherwithmaryafellowcatholicitwasapparenttothecourtthatmarywasafigureheadfortheconspiratorsbutit
wasnotclearthatshehadgivenherblessingtotheconspiracyinfactmaryhadauthorizedtheplotthechallengeforwal
singhamwastodemonstrateaclearlinkbetweenmaryandtheplottersonthemorningofhertrialmarysataloneinthedoc
kdressedinsorrowfulblackvelvetincasesoftreasontheaccusedwasforbiddencounselandwasnotpermittedtocallw
itnessesmarywasnotevenallowedsecretariestohelpherpreparehercasehoweverherplightwasnothopelessbecause
shehadbeencarefultoensurethatallhercorrespondencewiththeconspiratorshadbeenwritteninciphertheciphert
urnedherwordsintoameaninglessseriesofsymbolsandmarybelievedthatevenifwalsinghamhadcapturedtheletters
hecouldhavenoideaofthemeaningofthewordswithinthemiftheircontentswereamysterythentheletterscouldnotbe
usedasevidenceagainstherhoweverthisalldependedontheassumptionthathercipherhadnotbeenbrokenunfortunat
elyformarywalsinghamwasnotmerelyprincipalsecretarybutalsoenglandsspymasterhehadinterceptedmaryslette
rstotheplottersandheknewexactlywhomightbecapableofdecipheringthemthomasphelippeswasthenationsforemos
texpertonbreakingcodesandforyearshehadbeendecipheringthemessagesofthosewhoplottedagainstqueenelizabe
ththerebyprovidingtheevidenceneededtocondemnthemifhecoulddeciphertheincriminatinglettersbetweenmarya
ndtheconspiratorsthenherdeathwouldbeinevitableontheotherhandifmaryscipherwasstrongenoughtoconcealher
secretsthentherewasachancethatshemightsurvivenotforthefirsttimealifehungonthestrengthofacipher
```

2. Methodology

My **Python script** automates **substitution cipher decryption** by combining:

- **Frequency analysis**
- **Pattern-based refinement**
- **Parallel hill-climbing optimization**
- **Trigram-based scoring**

The decrypted text was evaluated using **trigram-based scoring**, which assigns **higher scores** to **decryptions resembling natural English** based on letter triplet frequencies.

A. **Frequency Analysis**

I first applied **letter frequency analysis**, mapping the most frequent **ciphertext letters** to common English letters (**e, t, a, o, etc.**), providing a **rough initial key mapping**.

```python
# 1) Frequency Analysis for Initial Key
def frequency_analysis_key(ciphertext):
    # Count frequency of each letter in ciphertext
    c_count = Counter(ch for ch in ciphertext.lower() if ch.isalpha())
    # Sort letters by frequency descending
    most_common_cipher_letters = [p[0] for p in c_count.most_common()]

    # If some letters never appear, add them to the end
    for alpha in letters:
        if alpha not in most_common_cipher_letters:
            most_common_cipher_letters.append(alpha)

    # Build mapping: ciphertext's highest freq letter -> 'e', 2nd -> 't', etc.
    # (english_freq_order is your best guess ordering for plaintext frequencies)
    key_map = {}
    for i, ciph_letter in enumerate(most_common_cipher_letters):
        if i < len(english_freq_order):
            key_map[ciph_letter] = english_freq_order[i]
        else:
            # If we run out of positions, map leftover letters arbitrarily
            key_map[ciph_letter] = random.choice(letters)

    return key_map
```

B. **Pattern Matching**

Next, I refined the mapping by **randomly swapping letters** and testing if the **trigram score improved**.

This step **corrected common words** like *"the"*, improving accuracy.

```python
# 2) Pattern Matching
def refine_with_pattern(ciphertext, key_map):
    """
    Example approach:
    - We look for bigrams/trigrams or a known word (e.g. "the") in the plaintext.
    - If substituting certain pairs of letters yields more occurrences of these words,
      that might be an improvement.
    """
    original_score = score_text(decrypt(ciphertext, key_map))
    best_map = dict(key_map)
    best_score = original_score

    # We'll attempt a small number of random letter swaps
    TRIES = 10000
    for _ in range(TRIES):
        # pick two random letters in the key_map
        a, b = random.sample(letters, 2)

        # swap their plaintext assignments
        cipher_for_a = None
        cipher_for_b = None
        for k, v in best_map.items():
            if v == a:
                cipher_for_a = k
            elif v == b:
                cipher_for_b = k

        if cipher_for_a is None or cipher_for_b is None:
            continue

        # Make a local copy for testing
        test_map = dict(best_map)
        test_map[cipher_for_a], test_map[cipher_for_b] = b, a

        test_score = score_text(decrypt(ciphertext, test_map))
        if test_score > best_score:
            best_score = test_score
            best_map = test_map

    return best_map
```

C. **Parallelized Refinement**

I then ran a **parallel refinement procedure**, where multiple processes:

1. **Performed additional swaps**
2. **Scored decryptions**
3. **Selected the best mapping**

```python
# 3) Parallel refinement worker function
def parallel_refine_worker(args):
    """
    Each worker:
    - Takes ciphertext + an initial key_map
    - Does some random swaps or small hill climbing
    - Returns best local result
    """
    ciphertext, initial_map, rounds = args
    best_map = dict(initial_map)
    best_score = score_text(decrypt(ciphertext, best_map))

    for _ in range(rounds):
        a, b = random.sample(letters, 2)

        cipher_for_a = None
        cipher_for_b = None
        for k, v in best_map.items():
            if v == a:
                cipher_for_a = k
            elif v == b:
                cipher_for_b = k

        if not cipher_for_a or not cipher_for_b or cipher_for_a == cipher_for_b:
            continue

        new_map = dict(best_map)
        new_map[cipher_for_a], new_map[cipher_for_b] = b, a

        new_score = score_text(decrypt(ciphertext, new_map))
        if new_score > best_score:
            best_map = new_map
            best_score = new_score

    return best_map, best_score
```

D. To evaluate decryption quality, I used **trigram-based scoring**. A **trigram** is a **three-letter sequence**, and its frequency in English helps determine how **realistic** a decrypted text appears.

```python
#sample text for trigram scoring
sample_text = """In the grand halls of history, civilizations have risen and fallen, leaving behind echoes of their achievements.
The written word has been the cornerstone of knowledge, preserving ideas across generations. From the philosophers
of ancient Greece to the scholars of the Renaissance, the pursuit of wisdom has been relentless.

Scientific discoveries have reshaped human existence. The understanding of gravity, the laws of motion, and
the structure of the atom have unlocked the mysteries of the universe. Medicine, once bound by superstition,
now thrives on the principles of biology and chemistry, extending human life beyond what was once imaginable.

Great minds such as Isaac Newton, Albert Einstein, and Marie Curie have illuminated the path of discovery,
challenging conventions and revolutionizing thought. Literature, too, has played its role in shaping society,
offering profound insights into the human condition. The words of Shakespeare, Austen, and Orwell
continue to inspire and provoke deep reflection.

Meanwhile, the industrial revolution transformed economies, leading humanity from agrarian societies to
technological marvels. The steam engine, electricity, and the advent of computers propelled civilization
into an era of unprecedented progress. Yet, with every leap forward, ethical dilemmas have emerged,
forcing society to confront the consequences of innovation.

The digital age has further accelerated change, bridging continents through instantaneous communication.
Artificial intelligence, once a concept of science fiction, now influences daily life, from healthcare to
autonomous vehicles. The balance between convenience and privacy has become a topic of global debate,
as technological advancements shape the future.

Despite the complexity of the modern world, fundamental values remain. The quest for truth, justice, and
understanding persists, uniting humanity in an ongoing narrative of growth and enlightenment.
Through the study of history, science, and literature, we continue to decode the past, navigate the present,
and anticipate the future.
"""

sample_text = sample_text.lower()
sample_text_clean = "".join(ch for ch in sample_text if ch.isalpha() or ch.isspace())
sample_text_joined = "".join(sample_text_clean.split())

def get_trigrams(text):
    trigrams = {}
    for i in range(len(text)-2):
        tri = text[i:i+3]
        trigrams[tri] = trigrams.get(tri, 0) + 1
    return trigrams

trigrams_sample = get_trigrams(sample_text_joined)
total_trigrams = sum(trigrams_sample.values())
trigram_probs = {
    tri: math.log(count / total_trigrams) for tri, count in trigrams_sample.items()
}
min_log_prob = math.log(1.0 / (total_trigrams * 100))  # penalty for unseen trigrams

def score_text(plaintext):
    pt = "".join(ch for ch in plaintext.lower() if ch.isalpha())
    s = 0
    for i in range(len(pt) - 2):
        tri = pt[i:i+3]
        s += trigram_probs.get(tri, min_log_prob)
    return s
```

3. Results & Observations

Through repeated attempts, the algorithm successfully derived the correct **encryption key** by first determining the decryption key and then inverting it. This allowed me to fully restore the original plaintext. Below is the final 100% accurate **plaintext letters to ciphertext letters (encryption key):**

```
=== Substitution Cipher Decryption Results ===
Best Overall Score: -28790.99559182848

Encryption key (plaintext to ciphertext): zqhlgmdfwcjnkovpeyxtrisbua
Decryption key (ciphertext to plaintext): zxjgqhecvkmdflnpbuwtyoisra
```

So far, in all test runs, the code has consistently found the correct decryption key within the automated five retries, demonstrating its **robustness** and **reliability**. This highlights the effectiveness of **stochastic search methods**, where **random swaps and trigram-based scoring** refine the key until an **optimal mapping** is found. Even when the first attempt is imperfect, **subsequent refinements consistently converge on the correct plaintext**.

4. Comparison with Lab 1

While Lab 1 also relied on **frequency analysis**, this coursework extends the approach with **trigram-based scoring** and **iterative refinement** instead of single-letter analysis. **Pattern matching** improves letter substitutions, identifying **common bigrams and trigrams**.

Additionally, **parallel processing** significantly accelerates decryption by running multiple hill-climbing attempts simultaneously. In contrast, Lab 1's **manual trial-and-error adjustments** were slower and less scalable. By combining **randomization, pattern detection, and parallelism**, this approach proves **more efficient and accurate**.

5. Conclusion

I successfully recovered the **encryption key** by first determining the **decryption key** and then inverting it. This was achieved through **frequency analysis, pattern matching, and parallel refinement**, ensuring **high accuracy**. Compared to Lab 1, my method leveraged **trigram-based scoring** and **parallel processing**, leading to **faster convergence and more reliable decryption**. This demonstrates how **randomization, statistical analysis, and computational efficiency** enhance **substitution cipher decryption**.