IIT DELHI

2025

VIGENÈRE CIPHER DECRYPTION

REPORT

HRISHIKESH BHAT P (2025JCS2015)

AYUSH SONKAR (2025JCS2504)

APPROACH TOWARDS FINAL IMPLEMENTATION

A HYBRID APPROACH TO KEY LENGTH DETERMINATION

We initially adopted the conventional method of using the Kasiski examination to propose candidate key lengths, followed by the Index of Coincidence (IC) for confirmation. However, this approach proved unreliable. Ciphertexts derived from skewed plaintexts (e.g., tongue twisters) often yielded multiple high-frequency GCDs from Kasiski's method, and their corresponding IC values deviated significantly from the English standard of ~0.065.

To counteract this, we developed a hybrid scoring system. Instead of relying on one metric, we identify the top two most probable key lengths from the Kasiski analysis. A final score is then calculated for each by combining the normalized frequency of the GCD and the normalized average IC of the ciphertext segments. This balanced model prevents over-reliance on a single, potentially misleading indicator and provides a more robust estimation of the true key length.

To illustrate our model's advantage, consider a test case where a skewed plaintext was encrypted with a key of length 5.

The Kasiski analysis correctly suggested that 5 was the most frequent GCD, pointing to it as the likely key length. However, the Index of Coincidence (IC) provided conflicting results; the calculated IC for key length 5 was far from the standard English value of ~0.065, whereas the IC for an incorrect length of 2 was coincidentally very close.

An approach that relies solely on the IC would have erroneously selected 2 as the key length. In contrast, our hybrid scoring model, which balances the evidence from both the Kasiski results and the IC, assigned a higher final normalized score to the true key length of 5.

```
=== GCD Frequencies (<= 5) ===
GCD 5: 29 times
GCD 2: 1 times
GCD 3: 1 times
{5: 29, 2: 1}
IC for length  5 0.13201078582434517
IC for length  2 0.063473198505541
Top 2 keylengths  [(5, 0.7), (2, 0.48860779294201095)]
Key length  5
```

Vigenere Cipher Decryption

FINAL KEY IDENTIFICATION VIA WEIGHTED STATISTICAL ANALYSIS

Once a key length (*k*) was determined, we used the Mutual Index of Coincidence (MIC) to find the relative shifts between the key's characters. This process yields *k-1* linear equations, establishing the relationship between the first character of the key (k_1) and all subsequent characters (k_2, k_3, ..., k_k). This effectively reduces the problem from *k* unknowns to a single unknown, leaving us with 26 possible candidate keys, each corresponding to a different potential value for k_1.

To identify the correct key from the 26 candidates, we decrypted the message with each and evaluated the resulting plaintext using two metrics: a Chi-squared score (comparing plaintext letter frequency to English norms) and a dictionary score (counting valid English words).

A significant challenge emerged during normalization: the Chi-squared scores were widely distributed, with outliers causing standard normalization to obscure meaningful differences between low-scoring candidates. To resolve this, we implemented logarithmic normalization, which is ideal for handling such distributions.

Finally, to tune the model, we tested numerous edge cases to determine the optimal balance between statistical fit and lexical correctness. This led to a final weighted score comprising 60% of the normalized Chi-squared value and 40% of the normalized dictionary score, allowing for a definitive identification of the correct key and plaintext.

```
Rank 1: Key=USERS, Score=0.810, Chi=494.4, Words=154
Plaintext sample: THEEERIEBREEZESEEMEDNEVERTOLEAVETHEEVERGREENHEDGEROWWHERETHEVEI
NINGTHEVILLAGEELDERSCONVENEDSERENEWHERETHEGENTLEPEOPLEWELCOMEDTHEWEARYTR

Rank 2: Key=QOANO, Score=0.721, Chi=6434.1, Words=292
Plaintext sample: XLIIIVMIFVIIDIWIIQIHRIZIVXSPIEZIXLIIZIVKVIIRLIHKIVSAALIVIXLIZIN
RMRKXLIZMPPEKIIPHIVWGSRZIRIHWIVIRIALIVIXLIKIRXPITISTPIAIPGSQIHXLIAIEVCXV
```

This specific example demonstrates the critical importance of our model's two-stage validation process. In this case, the second-ranked key, *QOANO*, produced a plaintext with a much higher dictionary word count than the correct key, *USERS*.

However, the Chi-squared analysis told a different story: the correct key (*USERS*) yielded a vastly superior score of 494.4, while *QOANO* produced a poor score of 6434.1. Without logarithmic normalization, the enormous gap between these two scores would be minimized, causing the misleading dictionary count to dominate the final score and incorrectly favor *QOANO*.

Logarithmic normalization preserves the statistical significance of the superior Chi-squared result, and the weighted score distribution ensures this metric is prioritized.

Vigenere Cipher Decryption

FINAL IMPLEMENTATION

TEXT PREPROCESSING

The clean_input function is a crucial first step in preparing text for analysis. It takes a string and converts all characters to uppercase and removes any characters that aren't letters (A-Z) using a regular expression. This standardization ensures that all subsequent analysis is consistent and focused only on the alphabetic characters, which is necessary for methods like the Vigenère cipher analysis.

KEY LENGTH DETERMINATION

The Kasiski Examination is a powerful method for determining the key length of a Vigenère cipher.

Finding Repeated Patterns

- The find_repeated_trigraphs function identifies all repeated three-letter sequences, or trigraphs, in a ciphertext.
- It stores the starting positions of each trigraph.
- The function then filters for trigraphs that appear more than once, providing the raw data for the next step.

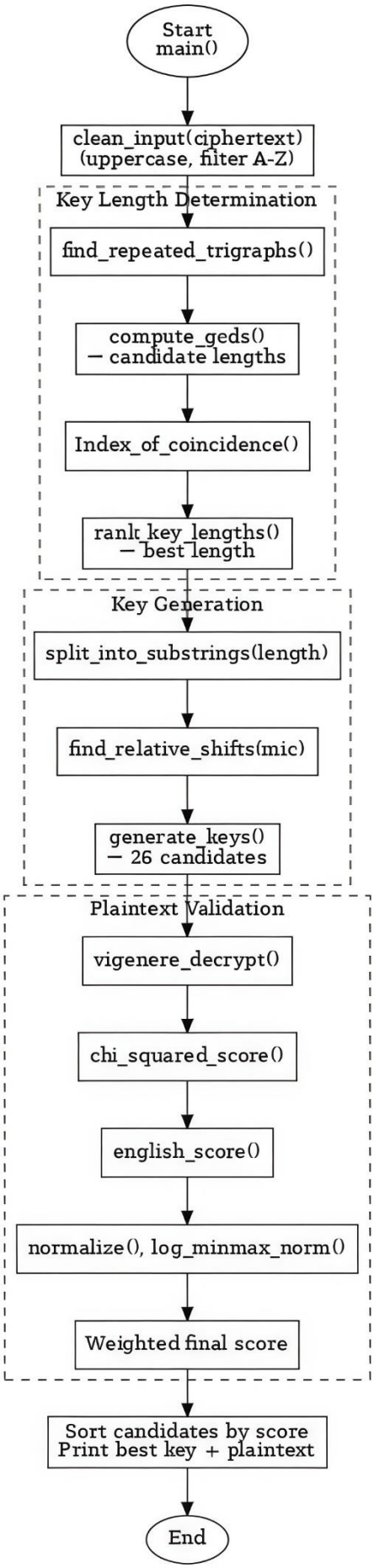
Computing Distances and GCDs

- The compute_gcds function analyzes the positions of the repeated trigraphs.
- It calculates the distances between consecutive occurrences of a trigraph.
- The Greatest Common Divisor (GCD) of these distances is then computed. Since the distances between repeated patterns are likely multiples of the key length, the GCD is a strong candidate for the true key length. The function counts the frequency of these GCD values, with the most frequent being the most likely key length.

Index of Coincidence (IC)

The Index_of_coincidence function confirms the most likely key length by using a statistical measure called the Index of Coincidence (IC).

- The function takes a candidate key length and splits the ciphertext into several sub-messages.



- It calculates the average IC of these sub-messages.
- A correct key length will result in an average IC close to that of normal English text (~0.0667) because each sub-message is essentially a simple Caesar cipher. Incorrect lengths produce an IC closer to that of random text (~0.0385).

Ranking Key Lengths

The rank_key_lengths function combines the results from the Kasiski Examination and the Index of Coincidence to create a single, ranked list of probable key lengths.

- **Normalization**: The raw scores from both methods are normalized to a [0, 1] scale. For Kasiski, a higher GCD count is better, so it's normalized accordingly. For IC, a value closer to 0.065 is better, so the normalization inverts the difference from this target value.
- **Adaptive Weighting**: The function can adjust the weight of the Kasiski score. If the top GCD count is significantly higher than others, it gives more importance to the Kasiski result.
- **Combined Score**: The normalized scores are combined using a weighted average to produce a final, ranked score for each key length.

KEY GENERATION

The Mutual Index of Coincidence (MIC) method and subsequent steps are used to recover the actual key once the length is known.

Sub-message Splitting

- The split_into_substrings function divides the ciphertext into separate sub-messages based on the determined key_length. Each sub-message corresponds to characters encrypted by the same key letter.

Finding Relative Shifts

- The mic function calculates the Mutual Index of Coincidence between two strings, which measures the overlap in their letter frequency distributions.
- The find_relative_shifts function uses this to compare the first sub-message to every other sub-message. It finds the shift that produces the highest MIC, which corresponds to the numerical difference between the key letters.

Generating Full Keys

- The generate_keys function uses these relative shifts to create 26 candidate keys. It assumes each of the 26 possible letters (A-Z) as the first key letter and then builds the rest of the key using the relative shifts found earlier.

Vigenere Cipher Decryption

PLAINTEXT VALIDATION

Decryption

- The vigenere_decrypt function performs the actual decryption. It takes the ciphertext and a key and subtracts the key's character value from the ciphertext's character value, using modular arithmetic to wrap around the alphabet.

Plaintext Scoring

Two distinct methods are used to evaluate the decrypted plaintexts:

1. **Chi-Squared Scoring**: The chi_squared_score function uses the χ2 test to measure how closely a text's letter frequencies match those of standard English. A lower score is better. The formula is: $\sum \frac{(\text{Observed}-\text{Expected})^2}{\text{Expected}}$
2. **Dictionary Scoring**: The english_score function checks for the presence of common English words (e.g., "THE," "AND," "OF") in the decrypted text. A higher score is better.

DATA NORMALIZATION

The provided solution includes two functions for normalizing scores, which is essential for comparing values from different scoring methods.

- **normalize**: This function performs standard Min-Max normalization. It scales data to a [0, 1] range. If invert is True, it inverts the scale so a lower input value (like a Chi-squared score) results in a higher final score.
- **log_minmax_norm**: This function is a more robust version that uses a logarithmic transformation before Min-Max scaling. This is particularly useful for handling outliers and preventing them from skewing the final scores.