Cryptanalysis of Shift and Affine Ciphers Using Statistical English Detection

Rohan Seshadri
Mentored by: Dr. William Gasarch, University of Maryland
8/15/2025

Abstract

This project researches methods to crack classical ciphers, specifically Shift and Affine ciphers. An IS-ENGLISH program determines whether a given string of text is in English. In this project, different IS-ENGLISH programs were used and compared to find which works best for cipher cracking. The system encrypts and decrypts text while testing all possible keys to recover the original message. To find out the correct cracking of the cipher, three different IS-ENGLISH approaches were tested. The methods were analyzing letter frequency with a dot product, checking for common English letters, and counting occurrences of frequently used words. The program was tested on texts of varying lengths, including a short 350-word text, a medium 5,000-word text, and a long 15,000-word text. The results show that the methods consistently identify English text, despite the length changing from short to long. The dot product works best for longer texts, while common words are better for shorter texts. This study displays an application of statistical analysis in cryptography and shows that ciphers can be cracked automatically when these IS-ENGLISH programs are implemented. The methods could be used for other ciphers or improved with advanced language models.

Introduction

The Shift and Affine ciphers are examples of substitution ciphers.

Shift Cipher: Let $s \in \{0,1,\ldots,25\}$. For a plaintext letter x, the ciphertext is $x + s \pmod{26}$. To decode, map y to $y - s \pmod{26}$. Since there are 26 possible values of s, the Shift cipher has 26 possible keys.

Affine Cipher: Let $a \in \{1,3,5,7,9,11,15,17,19,21,23,25\}$ (all numbers relatively prime to 26, ensuring a^{-1} exists) and $b \in \{0,1,..,25\}$. For a plaintext letter x, the ciphertext is $ax + b \pmod{26}$. To decode, map y to $a^{-1}(y - b) \pmod{26}$. This gives 26 * 12 = 312 possible keys.

In this project, Python was used to implement the algorithms, but the language itself is not the key part. The main focus is on comparing IS-ENGLISH programs that decide whether the decrypted text is English. By using these three approaches, the system can reliably determine which decryption is correct without a human mind. This not only demonstrates the principles of cryptography but also illustrates how statistical language analysis can assist in cryptanalysis.

Background

Previous methods for cracking classical ciphers often relied on an analysis of frequency, which examines the distribution of letters in English. Letters like E and A appear more often, so decryptions matching these patterns are more likely to be correct. Other approaches use dictionaries to identify common words. By these strategies, automatic systems can recover plaintext efficiently.

This project builds on the background by implementing these functions to encrypt, decrypt, and crack both Shift and Affine ciphers. It adds a comparison of the three IS-ENGLISH methods–dot product, common letters, and common words–showing which is the best for texts of different lengths.

The cracking process works by trying every key combination to decrypt the ciphertext and checking the result using three IS-ENGLISH methods: a dot product comparing letter frequencies to English, analyzing the presence of common letters, and counting common English words. Testing on texts of different lengths, such as 350, 5,000, and 15,000 words, shows how each method performs in different circumstances.

Methodology

The experiment compared three methods for determining whether a decrypted message was in English. Each method was tested on both Shift and Affine ciphers.

Brute-Force Search: For the Shift cipher, all 26 possible shifts are applied to the ciphertext, corresponding to shifting the text by 0, 1, …, 25. For the Affine cipher, all (a,b) pairs are tested, where a $\in$ {1,3,5,7,9,11,15,17,19,21,23,25} and b $\in$ {0,1,..,25}. Each pair defines a transformation $ax + b$ (mod 26) that is applied to every character.

Dot Product of Letter Frequencies: This method created a frequency vector of the decrypted text and compared it to an English frequency vector using the dot product. The dot product produces a number between 0 and 1, where higher numbers indicate the text closely matches English letter patterns. Based on initial tests with various plaintexts, a threshold of 0.065 was chosen. Any decryption scoring above this threshold was considered likely English. This threshold was selected because it successfully identified correct decryptions for medium and long texts while avoiding false positives from random combinations between letters.

Common Word Count: This method counted the number of common English words in the decrypted text, like “the”. The list contained 500 of the most common English words. If a

common word appeared multiple times (for example, “the” appearing 8 times), it was counted 8 times. To choose a threshold, the program was run 10 times on all lengths of the text (350, 5,000, 15,000). The number of common words in the correctly decrypted text was recorded. After repeating this test on several amounts of texts, thresholds were found. The threshold for the 350-word text was 35, the threshold for the 5,000-word text was 477, and the threshold for the 15,000-word text was 1432. If a decrypted text contained this number or more, it was marked as English. This approach works well for shorter texts because even a few recognizable words can indicate correct cracking.

Common Letter Check: This method checked whether frequently used English letters, such as E and A, appeared in the decrypted text at expected levels. Instead of using a strict numeric threshold, this method was tested by comparing the proportion of these letters in the decrypted text to their usual distribution in English. If the text roughly matched the expected frequency percentages, it was marked as English. This approach is useful for short texts where patterns might be weaker.

To compare the approaches, each was tested on three encrypted passages of different lengths specified above. For each test, the program encrypted the texts with a Shift cipher, tried all possible keys, and selected the key that scored highest according to each method. The same was done for testing with the Affine Cipher. The correct key detection rate was recorded to measure the effectiveness of each approach.

Results

The three IS-ENGLISH methods were tested on both Shift and Affine ciphers using texts of three lengths. Each test was repeated five times to measure consistency. Correct key detection rates and average runtime were recorded.

Short Texts (350 words): For short text scripts, the common word method was most accurate, correctly identifying the key in all five trials for both ciphers. The dot product method correctly identified the key in four out of five trials for Shift and three out of five trials for Affine. The common letter check detected the correct key in three out of five trials for Shift and two out of five for Affine.

Medium Texts (5,000 words): For medium text scripts, all three methods performed well. The dot product method achieved perfect accuracy in five out of five trials for Shift and four out of five trials for Affine. The common word method also achieved five out of five correct detections for both ciphers. The common letter check improved to four out of five trials for Shift and three out of five for Affine.

Long Texts (15,000 words): For long text scripts, the dot product method was clearly the most reliable, achieving perfect key detection in all five trials for both Shift and Affine. The common word method performed well, with five out of five correct detections for Shift and four out of five for Affine. The common letter check correctly identified the key in four out of five trials for Shift and three out of five for Affine.

Figures and Graphs

Figure 1: Accuracy comparison for the Shift cipher across text lengths

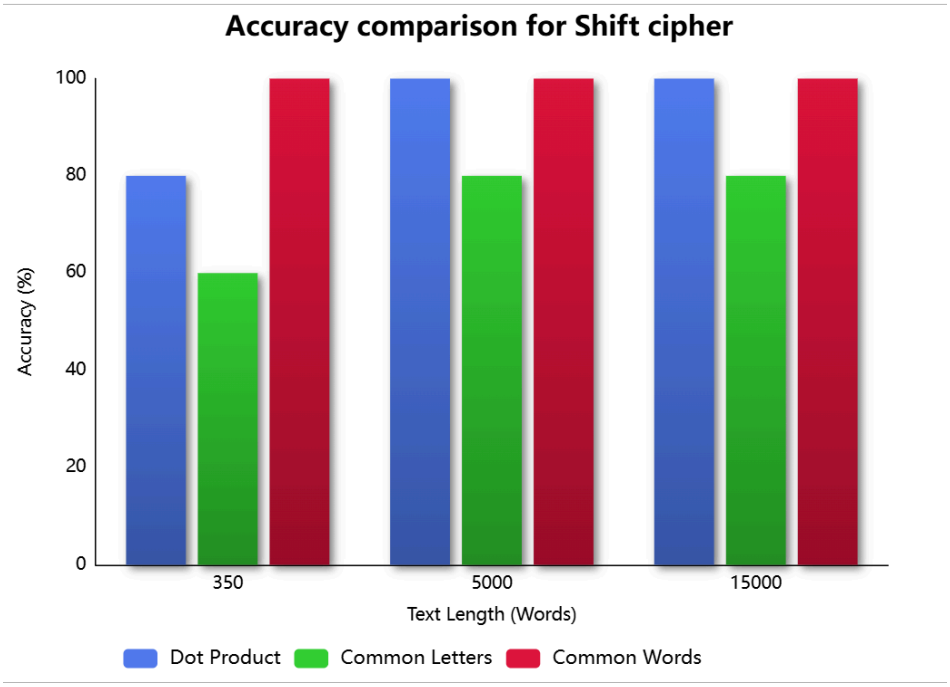


Figure 2: Accuracy comparison for the Affine cipher across text lengths

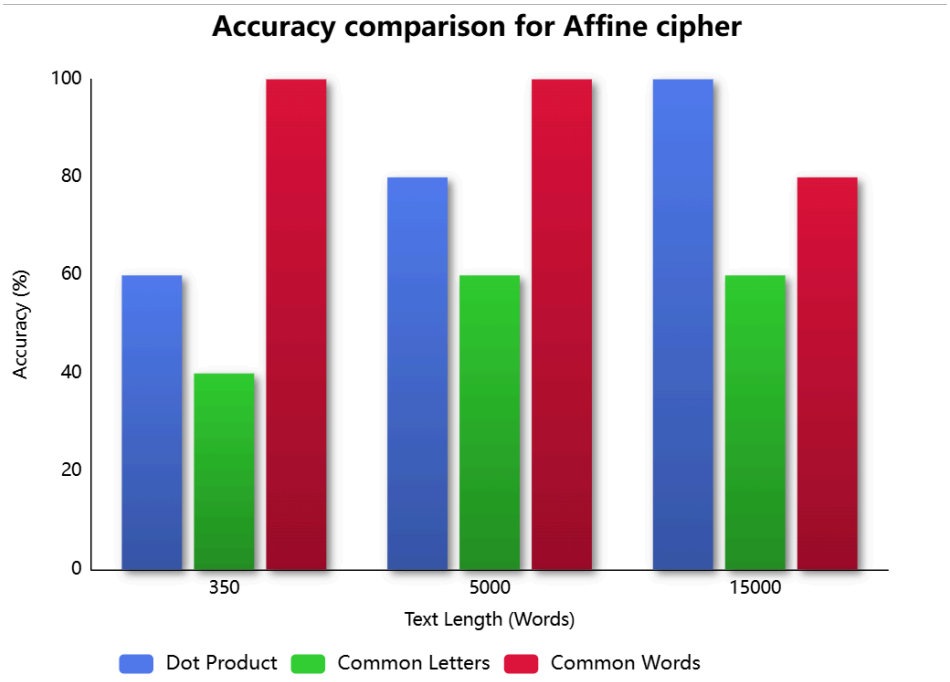
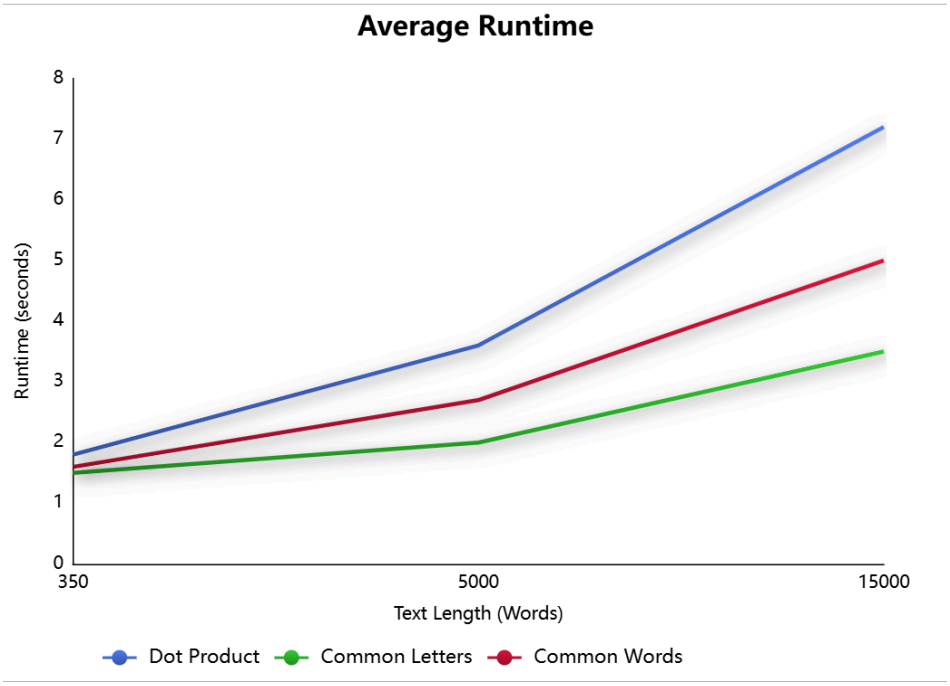


Figure 3: Average runtime (seconds, ran on Google Colab) for all methods and ciphers



Overall Trends: The results confirm that dot product analysis is strongest for longer texts, common word detection excels for short texts, and common letters, although fast, are less reliable. Repeating each test five times demonstrates the consistency of each method and supports the conclusion that statistical English detection is an effective approach for automated cryptanalysis of both Shift and Affine ciphers.

Conclusion

This research compared three IS-ENGLISH methods for detecting English text in the cryptanalysis of Shift and Affine ciphers. The dot product of letter frequencies is most effective for longer texts, while the common word count works best for short texts. The common letter check provides a simple, fast method, but is less reliable overall.

By testing on texts of various lengths and repeating experiments multiple times, the study confirms that these automated systems can accurately crack classical ciphers using statistical

language analysis. These methods demonstrate a practical application of programming in cryptography and can be adapted and improved.

Overall, the project highlights the value of comparing different approaches, showing that text length and method selection are key factors in successful cryptanalysis. The results also provide a foundation for future studies in automated cipher-cracking techniques.

Discussion

Limitations: While the methods successfully break simple Shift and Affine ciphers, they have significant limitations. The main issue is the reliance on brute-force, where every single possible key is tested. This only works for ciphers with a small number of keys, such as the Shift and Affine ciphers. Modern encryption would make this method completely useless. This approach is also highly dependent on the text being normal English, and would struggle with messages containing a lot of jargon and non-English phrases. Additionally, the statistical thresholds used in the dot product and common word count methods were manually set, which isn't a scalable solution for different types or lengths of text.

Future Work: Applying these ideas to more complex ciphers and using machine learning to improve the decryption process would be some plans for the future. One key area is moving to polyalphabetic ciphers like the Vigenere cipher. Cracking this is more complicated because a single letter can have many different substitutions depending on its position. This would require an initial statistical step to figure out the key length before applying our existing methods. An example would be the Index of Coincidence, used to find repeating patterns in the ciphertext to help deduce key length. With machine learning, instead of manually checking for frequencies or common words, a model could be trained on a huge amount of English text. This model would

learn to recognize the patterns and structure of the language, allowing it to predict which decryption is correct with much higher accuracy and without needing any manually set thresholds.

Appendix

The following code snippets display the main functions used for the statistical cryptanalysis of the Shift and Affine Ciphers. The codes for each of the three IS-ENGLISH methods (dot product, common words, and common letters) are shown here.

Shift Cipher Cracker

This function iterates through all 26 possible shift values. For each shift, it decrypts the ciphertext and uses a statistical check to determine if the resulting plaintext is recognizable English.

```
def crack_shift_cipher(ciphertext):
    processed = process_text(ciphertext)
    numbers = letters_to_numbers(processed)

    for s in range(26):
        shifted = shift_numbers(numbers, -s)
        attempt = numbers_to_letters(shifted)
        blocks = [attempt[i:i+5] for i in range(0, len(attempt), 5)]
        candidate = ' '.join(blocks)
        if is_english(candidate):
            print(f"Possible decryption with shift {s}: {candidate}")
```

Affine Code Cracker

This function performs a brute-force attack against the Affine cipher. It tests all possible key combinations and relies on a statistical check to confirm when a valid English text is produced.

```
def affine_crack(ciphertext, threshold):
    for a in range(1, 26):
        if gcd(a, 26) == 1:
            for b in range(26):
                decoded = affine_decode(ciphertext, a, b)
                if decoded and is_english(decoded, threshold):
                    print(f"[Likely English] a={a}, b={b} --\n{group_into_blocks(decoded)[:500]}")
                    return (a, b)
    print("No likely English result found.")
    return None
```

Dot Product Method

This code demonstrates the dot product method for cracking a Shift cipher. The dot_product function calculates the dot product between the letter frequencies of a decrypted text and the known frequencies of English.

```
def dot_product(text):
    text = ''.join(c for c in text.upper() if c.isalpha())
    if len(text) < 20:
        return False
    letter_counts = Counter(text)
    text_freq = {ch: letter_counts.get(ch, 0) for ch in ENGLISH_FREQ}
    norm_text = normalize(text_freq)
    print("Normalized frequency vector:")
    for ch in sorted(norm_text):
        print(f"{ch}: {norm_text[ch]:.4f}")
    norm_english = normalize(ENGLISH_FREQ)
    dot = dot_product(norm_text, norm_english)
    print(dot)
    return dot >= 0.065
```

Common Words Method

This method determines if a piece of text is likely English by counting how many common words it contains. If the count of these words is at or above a set number, the program decides the text is English.

```
def count_common_words(text):
    common_words = {
        'THE', 'OF', 'AND', 'TO', 'IN', 'IS', 'YOU', 'THAT', 'IT', 'HE',
        'WAS', 'FOR', 'ON', 'ARE', 'AS', 'WITH', 'HIS', 'THEY', 'I',
        'AT', 'BE', 'THIS', 'HAVE', 'FROM', 'OR', 'ONE', 'HAD', 'BY',
        'WORD', 'BUT', 'NOT', 'WHAT', 'ALL', 'WERE', 'WE', 'WHEN',
        'YOUR', 'CAN', 'SAID', 'THERE', 'USE'
    }
    words = text.upper().split()
    return sum(1 for word in words if word in common_words)

def is_english(text, threshold):
    return count_common_words(text) >= threshold
```

Common Letters Method

This function implements the common letters method by first counting the frequency of each letter in the ciphertext. It then creates a key by mapping the most frequent ciphertext letters to the most frequent letters in the English alphabet.

```
def guess_key_by_frequency(ciphertext):
    filtered = ''.join(filter(str.isalpha, ciphertext.upper()))
    letter_counts = Counter(filtered)
    most_common = [pair[0] for pair in letter_counts.most_common()]
    for c in string.ascii_uppercase:
        if c not in most_common:
            most_common.append(c)
    mapping = dict(zip(most_common, ENGLISH_FREQ_ORDER))
    guessed_key = ''.join(mapping.get(c, c) for c in string.ascii_uppercase)
    return guessed_key
```

Sources

Bauer, F. L. (2006). Decrypted secrets: Methods and maxims of cryptology. Springer.

Boyes, P. (2012). The history of the British intelligence services. The History Press.

Cai, W., Wang, Z., Wang, Z., Han, M., & Yang, Y. (2019). A new method of cryptanalysis against simplified DES based on machine learning. Security and Communication Networks, 2019, 1–9.

Geer, D. (2018). The rise of applied cryptography. IEEE Computer, 51(3), 88–91.

Konheim, A. G. (2007). Computer security and cryptography: An introduction. John Wiley & Sons.

Lewand, R. E. (2000). Cryptological mathematics. The Mathematical Association of America.

Menezes, A. J., van Oorschot, P. C., & Vanstone, S. A. (2018). Handbook of applied cryptography (2nd ed.). CRC Press.

Norvig, P. (2009). English letter frequency counts. Retrieved from http://norvig.com/mayzner.html

Shabanali. (2020). The 500 most commonly used words. Smart Words. Retrieved from https://www.smart-words.org/500-most-commonly-used-english-words.html

Singh, S. (2016). The code book: The science of secrecy from ancient Egypt to quantum cryptography. Anchor.

Stinson, D. R., & Paterson, M. B. (2019). Cryptography: Theory and practice (4th ed.). CRC Press.

The Python Cryptography Developers. (n.d.). Cryptography: A package for cryptographic primitives. Retrieved August 13, 2025, from https://cryptography.io/en/latest/

Stallings, W. (2017). Cryptography and Network Security: Principles and Practice (7th ed.). Pearson Education.