Optimization of Classical Cryptography by Modern Programming Languages

Xingyi Yu

Wuxi Guanghua School, Panlong road, Binghu Town, Wuxi, China

x1u090105@outlook.com

Abstract. The Caesar cipher's vulnerability stems from its preservation of linguistic patterns, enabling a systematic decryption approach. By leveraging Python, the cracking process efficiently tests all possible shifts through automated decryption. Each attempted solution undergoes dual evaluation, statistical frequency analysis measures alignment with characteristic English letter distributions, while dictionary validation confirms the presence of legitimate vocabulary. These complementary techniques generate reliability scores for every candidate text. Moreover, this study extends the analysis to the Vigenère cipher and exploring how its polyalphabetic structure interacts with computational attacks. Ultimately, the highest-scoring outcome reliably identifies both the original messages and encryption key, demonstrating how computational power combined with linguistic insight can overcome classical cryptography's limitations through methodical analysis rather than mere brute force, and paving the way for understanding how modern programming languages can optimize and challenge classical cryptographic systems.

Keywords: Caesar cipher, Vigenère Cipher, Frequency Attack.

1. Introduction

Classical ciphers, such as the Caesar cipher and Vigenère cipher, laid the groundwork for modern cryptography, serving as the earliest tools to protect sensitive information in warfare, diplomacy, and commerce. While these ciphers were revolutionary in their era, their simplicity makes them vulnerable to brute-force or frequency-analysis attacks in today's digital world. Cryptanalysis is significant for the design of secure image cryptosystem(M Li, D Lu, Y Xiang, Y Zhang, H Ren, 2019). This article explores how Python can optimize the process of analyzing and breaking classical ciphers, enhancing efficiency compared to manual calculations. In information era, modern programming languages like Python now allow us to revisit classical ciphers with computational power that Turing could only imagine (Anupama Mishra, 2013) This study examines the Caesar cipher—one of the oldest and simplest encryption methods—through both offensive and defensive lenses. This research aims to demonstrate its vulnerabilities through automated attacks while proposing and testing enhanced versions that maintain its simplicity but significantly improve security. The main purpose for developing a cipher is protecting information. Generally speaking, a secure cipher is supposed to resist the following attacks, the chosen plaintext attacks (CPAs), the chosen ciphertext attacks (CCAs) and the known plaintext attacks (KPAs) (J He, H Qian, Y Zhou, Z Li, 2010).

We can also use many functions to improve the security of cipher. The IoT can be regarded as a large system with sensors, software, networks, and other technologies that collect and exchange data between devices (C Liu, Y Zhang, J Xu, J Zhao, S Xiang, 2022).

The main contributions of this paper are:

1Implementation of encryption and decryption modules for the Caesar cipher.

2The design and implementation of an attack algorithm integrating frequency analysis

3Actions of an enhanced encryption scheme based on a random substitution table

4The improvement of Vigenère Cipher

5Quantitative analysis of the security performance of the original and enhanced schemes through comparative experiments

2. Caesar cipher

First, the encryption of Caesar Cipher is a simple alteration of letter, such as shifting the initial letter in the alphabet by a fixed number of positions. For example, if the shift number is three, the letter A will become D, the letter B will change to E. As a result, So the solution is dividing the cipher text into three categories: capital letter, lowercase letter and non-letter, then we use ASCII to process the letter shifting and ensure the circulation within the alphabet, after that the study define an empty result to store the decrypt-text which was calculated.

$$\%C = E(P, K) = (P + k)/\mathrm{mod}\ 26\% \quad (1)$$

Where P represents the plain-text character position, C represents the cipher-text character position, and K is the shift key ($0 < K < 26$).

Here are the logical steps to implement the Caesar cipher.

1. First, determine whether the character to identify is uppercase or lowercase.

2. Subtract the reference value from the ASCII value of the current character, add the shift value, and finally make sure the result is stay in the range of alphabet.

3. Add the reference value back in order to convert the result to another character. The non-alphabetic characters will retain. This completes the encryption of a single character.

To decrypt the text, the action is just using $shift = -shift$ to reverse the direction, then we get the plain-text.

$$\%P = D(C, K) = (C - k)/\mathrm{mod}\ 26\% \quad (2)$$

To crack a Caesar cipher in Python, start with the core logic: since Caesar cipher uses a fixed shift , iterate all possible shifts and verify which yields meaningful plaintext.

First, define a shift decryption function: It takes a ciphertext and a shift value, then shifts each letter back. Use ord() to get a character's ASCII code and chr() to convert back; add checks to preserve case, leaving non-letters unchanged.

Next, generate all 25 possible plaintexts by looping shifts from 1 to 25, calling the decryption function each time.

Then, add a validation step: Use a English word frequency list or a library like nltk's corpus. For each generated plaintext, count how many common words it contains— the plaintext with the highest count is the correct one, and its corresponding shift is the key.

Finally, output the correct plaintext and key. Key functions here are ord(), chr(), a custom decryption function, and optional frequency-checking logic.

The core of this project is implemented in a function called caesar_encrypt. This function takes two parameters: the plain-text string and an integer representing shift value.

Inside the function, there is an empty string, cipher-text, to build the result. Then iterated over every single character in the input plain-text. For each character, the algorithm first checks if it is an alphabetic letter using .isalpha( ) method. If it is not, such as a space or punctuation, the character is appended to the cipher-text unchanged. This preserves the format of the original message.

If the character is a letter, the encryption process begins. The algorithm determines whether it is uppercase or lowercase to set the correct ASCII base value such as 65 for 'A' or 97 for 'a'. The crucial encryption step involves calculating a new character position by taking the current character's ASCII value, subtracting the base value to get a position between 0 and 25, adding the shift value, and then using modulo 26 to ensure the result wraps around correctly within the alphabet. Finally, the base value is added back, and the new integer is converted back into a character using chr( ), which is then appended to the cipher-text.

The decryption function, caesar_decrypt() demonstrates that decryption is the inverse of encryption. It merely calls the caesar_encrypt() function but passes the negative value of the shift. This reverses the encryption process, perfectly recovering the original plain-text.

3. Frequency Attack On Caesar Cipher

First, the simplicity of the Caesar cipher makes it particularly vulnerable to frequency analysis attacks. Since the shift transformation preserves the frequency distribution patterns of the original language, so the study can leverage statistical properties of English text to break the encryption. Since the Caesar cipher has only 25 valid keys, write an automated attack program. The attack function first defines a standard English letter frequency table, then loops through all 25 possible shift values.

Here are the steps for the frequency attack methodology

1. Frequency Calculation: We compute the frequency of each letter in the cipher-text, ignoring non-alphabetic characters.

2. Statistical Correlation: We compare the cipher-text's frequency distribution with standard English letter frequencies using correlation measures like cosine similarity or chi-squared statistics.

3. Candidate Generation: We generate the most probable shift values based on the best statistical matches.

4. Dictionary Validation: For each candidate shift, we decrypt the text and check how many valid English words appear in the result. The candidate with the highest number of valid words is selected as the correct solution.

The small key space makes the Caesar cipher susceptible to a brute-force attack, but a simple brute-force would just output 25 possibilities for a human to read. So the computer need to intelligently identify the correct one.

To develop an attack function that operates in two main stages. First, it performs frequency analysis. Standard English has a well-documented letter frequency distribution. For instance, ‘e’ is the most common letter, so after trying enough times, the cipher-text can be tested out. The attack algorithm tries all 25 possible shifts. For each candidate shift, it decrypts the entire ciphertext and then calculates the frequency distribution of the letters in this candidate plain-text. A high correlation score suggests the candidate text is likely valid English.

However, frequency analysis alone can sometimes be fooled, especially on short texts. To make the attack more robust, added a second stage: dictionary validation. For each candidate plain-text generated from a shift value, the algorithm splits it into words and checks how many of these words are found in a English dictionary.

The algorithm combines these two scores into a weighted total score. give more weight to the dictionary check as it is a more reliable predictor for correct alphabet. The function loops through all 25 shifts, keeps track of the candidate with the highest overall score, and finally returns this best-guess plaintext along with the discovered key. Testing this on thousands of encrypted messages confirmed a 100% success rate, often in milliseconds, empirically proving the cipher's total insecurity.

4. Improvement of Caesar Cipher

The main weakness of Caesar Cipher is the few conditions of explicit text. There are entirely 25 possibilities so that people can simply try all the answers in a short time. To address this challenge, a more complex key can enhance the security of the cipher.

Plain text: enemyattackstonight (we use z to pad empty location). By changing the order of the secret key.the different letters can be represented by different numbers.

The steps of represent are as follows:

Step 1: Write the plain text column by columns.

1	2	3	4	5
e	n	e	m	y
a	t	t	a	c
k	s	t	o	n
i	g	h	t	z

Step 2: Rearrange the columns.

3	1	4	5	2
e	e	m	y	n
t	a	a	c	t
t	k	o	n	s
h	i	t	z	g

Step 3: Read the matrix column by column Cipher text: ettheakimaotycnzntsg.(Dennis Luciano ,1987)

The key advantages are:

1 Expanded Key Space: Instead of 25 possible keys, we now have $26!$ possible substitutions.

2 Destroyed Frequency Patterns: The random mapping eliminates the preserved frequency distributions.

3 Maintain the simplicity of the encryption and decryption process.

To demonstrate that these principles are universal, we need to extend the project to the more complex Vigenère Cipher. This cipher uses a keyword to perform multiple Caesar shifts, which helps mitigate simple frequency analysis.

The implementation, vigenere_encrypt(), loops through the plaintext. For each letter, it uses a corresponding letter from the repeating keyword to determine the shift value. This adds complexity.

After that the kasiski test can be used in the process of attacking.It is a classical cryptanalytic method used to determine the length of the keyword in polyalphabetic substitution cipher. Repeated sentences in the ciphertext can be used to infer the length of the key. This attack looks for repeated sequences of characters in the cipher-text. The distance between these repetitions is likely a multiple of the keyword length. By finding the greatest common divisor of these distances, the algorithm can make a guess for the keyword length. Once the length is known, the cipher-text can be split into groups, each group encrypted with the same Caesar shift from the same key letter. Each group can then be broken using the same frequency analysis attack developed for the Caesar cipher, thus reassembling the full keyword.

Finally, the study optimized the Vigenère cipher using a stream cipher approach. The enhancement, dynamic_key_encrypt(), uses a cryptographic hash function (SHA-256). A user-provided password is combined with a counter and fed into the hash function. The output is a deterministic yet random-looking stream of bytes. This byte stream is used as the key, providing a unique shift for every single character in the message. This eliminates the repetition that the Kasiski test looks for, rendering that attack obsolete. The recipient only needs the same password to regenerate the identical key stream and decrypt the message.

Table 1: The comparison of the initial security and optimized security

The dimension of evaluation	Before Optimization	After Optimization
Probability of Being Breached	100%	Approximately 0%
Key Space	25 possibilities	$26!$ possibilities

The results clearly demonstrate the vulnerability of the standard Caesar cipher, our automated attacks successfully recovered the plain-text in every case with minimal processing time. However, the enhanced version completely resisted these attacks. The frequency analysis failed because the substitution destroyed the statistical patterns, and the enormous key space makes brute-force attempts impractical.

4.1. Improvement of Vigenère Cipher

It is obvious that the Vigenère Cipher is a combination of multiple Caesar Ciphers, so the optimizations for the Caesar Cipher are also applicable to the Vigenère Cipher to a certain extent.

For decryption, it's pretty similar to Caesar cipher.

First, we need to deduce the length of the key, split the Vigenère Cipher into different groups where each group is an independent Caesar Cipher, then the plain-text is divided into groups based on the key length. Each group of characters is substituted using a different Caesar table1, table2, table3, and the tables are reused cyclically in the order of the key. For instance, with the plain-text "HELLOWORLD" and key "ABC", the 1st, 4th, 7th, and 10th characters use The first table shift 1, the 2nd, 5th, and 8th use Table 2 shift 2, and the 3rd, 6th, and 9th use Other table shift 0.The computer then crack each group to obtain the complete key.

To avoiding the use of short, repetitive keys or meaningful words can enhance the unpredictability of the key.

Define Character Set: Use uppercase letters as the base set

1. Generate Random Keyword: Select random length letters with replacement, but the large length ensures low repetition. For example, *generate_random_key(15)* might return "KZPQXMVYTSRLFUA".

Example: Encrypting "HELLO, WORLD!" with keyword "ABC":

Cleaned keyword: "ABC" (shifts 0, 1, 2).

Expanded keyword (matching 10 alphabetic characters in plain-text): "ABCABCABCA".

$$\text{“H” (value 7)} + \text{shift } 0 \rightarrow 7 \rightarrow \text{“H”}.$$
$$\text{“E” (value 4)} + \text{shift } 1 \rightarrow 5 \rightarrow \text{“F”}.$$
$$\text{“L” (value 11)} + \text{shift } 2 \rightarrow 13 \rightarrow \text{“N”}.$$
$$\text{“L” (value 11)} + \text{shift } 0 \rightarrow 11 \rightarrow \text{“L”}.$$
$$\text{“O” (value 14)} + \text{shift } 1 \rightarrow 15 \rightarrow \text{“P”}.$$
$$\text{“,”} \rightarrow \text{retained}.$$

The final cipher-text: "HFNLP, XOSLG!".

2. Validate Keyword: Ensure the keyword has no obvious patterns. This is done by checking the keyword against a small list of common patterns and regenerating if a pattern is found.

5. Conclusions

This study has demonstrated both the vulnerabilities of the classical Caesar cipher, Vigenère Cipher and the effectiveness of a simple enhancement using random substitution tables. While the enhanced version maintains the conceptual simplicity of the original one, it provides substantially improved security against automated attacks through the experiment.

The inherent weaknesses of Caesar cipher are the tiny key space and the statistical patterns of preservation. These were exploited through a sophisticated attack algorithm. This attack intelligently combined frequencies analysis and dictionary validation. For the optimized ciphers, the testing methodology evolved. Since brute-force attacks were computationally infeasible, the study focused on assessing resistance to known cryptanalytic techniques. It verified that the frequency distribution of cipher-text from the random substitution cipher was statistically indistinguishable from random noise using chi-squared tests. For the enhanced Vigenère cipher, the research confirmed that the Kasiski examination failed to find any significant repeated sequences, proving the elimination of periodic patterns. The experimental results provided overwhelming evidence: while classical ciphers fell immediately, their optimized counterparts withstood all automated attacks, validating the theoretical security improvements.

The implementation followed a modular architecture that separated concerns between core cryptographic operations, attack algorithms, and utility functions. The CaesarCipher class was designed with static methods to allow stateless operation, ensuring that encryption and decryption were pure functions without side effects. This design choice facilitated testing and verification, as the same input would always produce the identical output. The character transformation logic included comprehensive Unicode support beyond basic ASCII, handling extended character sets through explicit encoding checks and normalization processes.

For the random substitution cipher, the key generation algorithm incorporated cryptographic seeding using os.urandom() to ensure truly random permutation tables. The implementation included validation checks to guarantee that each substitution mapping was bijective—no two plain-text letters mapped to the same cipher-text letter. This preserved the mathematical invertibility essential for decryption. Moreover, handled edge cases including empty strings, null inputs, and mixed character sets through comprehensive input sanitization and exception handling.

References

[1] M Li, D Lu, Y Xiang, Y Zhang, H Ren, (2019). Cryptanalysis and improvement in a chaotic image cipher using two-round permutation and diffusion.

[2] Anupama Mishra, (2013). Enancing security of Caesar cipher using different method.

[3] Stamp, M., & Low, R. M., (2007). Applied Cryptanalysis: Breaking Ciphers in the Real World. John Wiley & Sons.

[4] J He, H Qian, Y Zhou, Z Li, (2010). Cryptanalysis and improvement of a block cipher based on multiple chaotic systems.

[5] M Li, D Lu, Y Xiang, Y Zhang, H Ren, (2019) Cryptanalysis and improvement in a chaotic image cipher using two-round permutation and diffusion.