7 Caesar Cipher

Jacques Mock Schindler

10.09.2025

In cryptography, encryption is the process of transforming information in a way that, ideally, only authorized parties can decode. This process converts the original representation of the information, known as plaintext, into an alternative form known as ciphertext. Despite its goal, encryption does not itself prevent interference but denies the intelligible content to a would-be interceptor.
From Wikipedia

The first example of a cipher we will look at is the Caesar cipher.

The Caesar cipher is a simple substitution encryption technique in which each letter of the text to be encrypted is replaced by a letter a fixed number of positions away in the alphabet. For example, using a right letter shift of four, `A` would be replaced by `E`, and the word `CIPHER` would become `GMTLIV`. The technique is named after Julius Caesar, who used it in his letters. The simplicity of the Caesar cipher makes it a popular source for recreational cryptograms.
From Encyclopedia Britannica

Python Implementation

To implement the Caesar cipher in Python, we build on the string methods offered by Python. The `ord` function returns the Unicode code point for a given character, and the `chr` function returns the character that corresponds to a given Unicode code point.

```
1 # What is the Unicode code point of the character 'A'?
```

Simple (naïve) Implementation

First, we define a function that encrypts a given plaintext by shifting each letter by a specified number of positions in the alphabet.

```
1 plain = "CIPHER"
2 shift = 4
```

```python
def caesar_encode(plain, shift):
    cipher = ""

    for char in plain:
        shifted = ord(char) + shift
        cipher += chr(shifted)

    return cipher
```

Enhancded Implementation

But what happens if we reach the end of the alphabet? For example, if we shift `Z` by 4 positions, we would go past `Z`. To handle this, we can use the modulo operator `%` to wrap around the alphabet. The modulo operator gives the remainder of a division operation, which allows us to "wrap around" when we exceed the length of the alphabet. In the case of the Caesar cipher, we can use it to ensure that our shifted positions stay within the bounds of the alphabet.

Therefore, we will use modulo 26 (the number of letters in the English alphabet) to ensure that our shifted positions wrap around correctly. For example, if we shift `Z` by 4 positions, we would end up at `D`. This wrapping behavior is essential for the Caesar cipher to function correctly.
The calculation for the new position of a letter can be expressed as:

$$x' = (x + n) \mod 26$$

or, in another notation:

$$x' = x \oplus_{26} n$$

Let's implement this in Python:

```python
def caesar_encrypt_mod(plain, shift):
    cipher = ""

    for char in plain:
        shifted = (ord(char) - ord('A') + shift) % 26 + ord('A')
        cipher += chr(shifted)

    return cipher
```

Because

```
ord('A')
```

returns 65, we need to subtract 65 from the result of `ord('A')` to get 0. This gives us the 0-based index of the letter in the alphabet, which is useful for our calculations.

Hence the calculation shown above.

For the decryption, we can simply subtract the shift value instead of adding it:

```
def caesar_decrypt_mod(plain, shift):
    cipher = ""

    for char in plain:
        shifted = (ord(char) - ord('A') - shift) % 26 + ord('A')
        cipher += chr(shifted)

    return cipher
```

For convenience, we implement one function for both encryption and decryption.

```
def caesar(text : str, shift : int, encrypt=True) -> str:
    text = text.upper()
    result = ""

    if encrypt:
        for char in text:
            shifted = (ord(char) - ord('A') + shift) % 26 + ord('A')
            result += chr(shifted)
    else:
        for char in text:
            shifted = (ord(char) - ord('A') - shift) % 26 + ord('A')
            result += chr(shifted)

    return result
```

Breaking Caesar Cipher

There are two main methods to break a Caesar cipher: 1. Brute Force Attack: Try all possible shifts (1-25) and see which one produces a meaningful result. 2. Frequency Analysis: Analyze the frequency of letters in the ciphertext and compare it to the expected frequency of letters in the language.

Brute Force Attack

```python
def caesar_bruteforce(ciphertext: str) -> None:
    for shift in range(1, 26):
        decrypted_text = caesar(ciphertext, shift, encrypt=False)
        print(f"Shift {shift}: {decrypted_text}")
```

Frequency Analysis

In the English language, certain letters appear more frequently than others. For example, the letter 'E' is the most common letter in English text, followed by 'T', 'A', 'O', 'I', and 'N'. By analyzing the frequency of letters in the ciphertext, we can make educated guesses about which letters correspond to which in the plaintext.

For more accurate analysis, see the following frequency table:

Letter	Frequency
E	12.02%
T	9.10%
A	8.12%
O	7.68%
I	7.31%
N	6.95%
S	6.28%
R	6.02%
H	5.92%
D	4.32%
L	4.03%
C	2.78%
U	2.76%
M	2.41%
W	2.36%
F	2.23%
G	2.02%
Y	1.97%
P	1.93%
B	1.49%
V	0.98%
K	0.77%
J	0.15%
X	0.15%
Q	0.10%
Z	0.07%

Therefore, we need a function that counts the frequency of each letter in the ciphertext. The result should be a dictionary where the keys are the letters and the values are the counts of each letter. Or even better, the frequencies in percent.

To keep things simple, we assume that the input text only contains uppercase letters from `A` to `Z` and no spaces or punctuation.

```
def letter_frequency(text: str) -> dict:
    frequency = {}
    total_letters = 0

    for char in text:
        if char not in frequency:
            frequency[char] = 1
        else:
            frequency[char] += 1
        total_letters += 1

    for key, value in frequency.items():
        frequency[key] = (value / total_letters) * 100


    return frequency
```

After finding the most frequent letter in the ciphertext, we can assume that it corresponds to the letter ‘E’ in the plaintext. By calculating the shift needed to convert the most frequent letter to ‘E’, we can then decrypt the entire ciphertext using that shift value.

```
def find_shift(char: str, e = 'E') -> int:
    return (ord(char) - ord(e)) % 26
```

After guessing the shift, we can use our existing `caesar` function to decrypt the ciphertext. We can call the function with the encrypted text and the guessed shift value to obtain the original plaintext.

If needed, the implementation of the `find_shift` function can exchange the default value for the most frequent letter to another letter.