Caesar Cipher

David W. Agler

November 23, 2023

Caesar Cipher

The Caesar cipher is a method of encryption. It works by taking a message (plaintext) and substituting (in a specific way) each letter in the plaintext with another letter in the alphabet (ciphertext). The method of substitution is such that each letter in the alphabet of the plaintext is moved (or "shifted") a number of places left or right in the alphabet. For example, we might take the plaintext "AID", shift the alphabet to the right 3 places, and then replace each letter in "AID" with a letter 3 places to the right in the alphabet. So, "A" would be replaced with "D", the letter "I" would be replaced with the letter "L", and the letter "D" would be replaced with the letter "G". The result would be the ciphertext "DLG".

Plaintext	A	I	D
Ciphertext	D	L	G

The number of positions down the alphabet that we shift each letter is called the "shift" or "key". So, in the example above, the shift is 3. The shift can be any number from 1 to 25. For the plaintext "ABC" and a shift of 1, then the letter "A" would be replaced by the letter "B", the letter "B" would be replaced by the letter "C". And, at the end of the alphabet, the letter "X" would be replaced by the letter "Y", the letter "Y" would be replaced by the letter "Z", and the letter "Z" would be replaced by the letter "A".

Plaintext	A	B	C	.	X	Y	Z
Ciphertext (shift 1)	B	C	D	.	Y	Z	Z

The shift can be any number, other than 0 or the size of the alphabet. A shift of 0 is not shift at all. If the alphabet of the plaintext is 26, then a shift of 26 is also not a shift at all since the cipher text would be the same as the plain text.

Strengths and Weaknesses of the Caesar Cipher

One strength of the Caesar cipher is that it is very easy to understand and use. In addition, those unfamiliar with the basics of cryptography will not know how to decrypt the message. As such, it provides a very minimal level of security from friends with no knowledge of encryption or children.

There are many weaknesses of the Caesar cipher. First, it is one of the most well-known ciphers and so anyone with any knowledge of ciphers will recognize it. Second, it is very easy to break. There are only 25 possible shifts, so one way to break the code is by brute force. Namely, one can simply try all possible shifts. For example, suppose we have the cipher text "ITT LWOA IZM PIXXG." We can decrypt this message by considering all 25 possible shifts:

- Key 0: ITT LWOA IZM PIXXG.
- Key 1: HSS KVNZ HYL OHWWF.
- Key 2: GRR JUMY GXK NGVVE.
- Key 3: FQQ ITLX FWJ MFUUD.
- Key 4: EPP HSKW EVI LETTC.
- Key 5: DOO GRJV DUH KDSSB.
- Key 6: CNN FQIU CTG JCRRA.
- Key 7: BMM EPHT BSF IBQQZ.
- Key 8: ALL DOGS ARE HAPPY.
- Key 9: ZKK CNFR ZQD GZOOX.
- Key 10: YJJ BMEQ YPC FYNNW.
- Key 11: XII ALDP XOB EXMMV.
- Key 12: WHH ZKCO WNA DWLLU.
- Key 13: VGG YJBN VMZ CVKKT.
- Key 14: UFF XIAM ULY BUJJS.
- Key 15: TEE WHZL TKX ATIIR.
- Key 16: SDD VGYK SJW ZSHHQ.
- Key 17: RCC UFXJ RIV YRGGP.
- Key 18: QBB TEWI QHU XQFFO.
- Key 19: PAA SDVH PGT WPEEN.
- Key 20: OZZ RCUG OFS VODDM.
- Key 21: NYY QBTF NER UNCCL.
- Key 22: MXX PASE MDQ TMBBK.
- Key 23: LWW OZRD LCP SLAAJ.
- Key 24: KVV NYQC KBO RKZZI.
- Key 25: JUU MXPB JAN QJYYH.

Notice that the only key that produces a message that makes sense is key #8. So, we can conclude that the shift is 8 and the plain text is "ALL DOGS ARE HAPPY."

Third, the Caesar cipher cannot be

Python Implementation

In this section, we consider a Python implementation of the Caesar cipher. We will first create a function that takes a string, a key (the shift) and returns the encrypted string. This is our `caesar_encrypt` function. Next, we will create a function that does the opposite. Namely, it takes a string, a key (the shift), and returns the decrypted string (plaintext). This is our `caesar_decrypt` function. Finally, we'll look at how to "hack" or "crack" or "break" the Caesar cipher. We will first create a function that takes a string and returns a list of all possible shifts and the encrypted string for each shift.

Caesar Cipher Encryption

Let's begin by considering how to encrypt a plaintext message using the Caesar cipher. We'll start simple by creating a long string that contains the symbols of our alphabet. The function takes a string and a key (the shift) and returns the encrypted string.

```
SYMBOLS = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
```

Next, we'll create a function that takes two arguments and return one value. First, it should take our `plaintext` string. Second, it should take a key (the shift). Third, it should returns the encrypted string. We'll call this function `caesar_encrypt`. Let's setup the function, specify that it takes two arguments `plaintext` and `key`, declare a variable `ciphertext` and have the function return the `ciphertext`.

```
def caesar_encrypt(plaintext, key):
    ciphertext = ""
    return ciphertext
```

At this point, if we call our function, it will only return the empty string. What we want to do is loop through each character in `plaintext`, check its position in the `SYMBOLS`, let's define this as the `pos` and then add the key (shift) number to the `pos`. The result will give us the corresponding char in the `SYMBOLS`. To do this, we'll use a `for` loop over the `plaintext`, check to see whether the character is an alphabet character (for English: A-Z, upper or lower case). If it is, we'll define a variable `pos` that is equal to the position of the character in the `SYMBOLS` string. Since the character may be lower case, we'll convert it to upper case using the `upper()` method. We'll then return the ciphertext.

```
SYMBOLS = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'

def caesar_encrypt(plaintext, key):
    ciphertext = ""
    for char in plaintext:
        if char.isalpha():
            pos = SYMBOLS.find(char.upper())
```

```
            pos = pos + key
    return ciphertext
```

Our function still will return a blank string, but now that we have the index of our new `char`, we only need to get that `char` from the `SYMBOLS` string and add it to the `ciphertext`.

```
SYMBOLS = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'

def caesar_encrypt(plaintext, key):
    ciphertext = ""
    for char in plaintext:
        if char.isalpha():
            pos = SYMBOLS.find(char.upper())
            pos = pos + key
        ciphertext = ciphertext + SYMBOLS[pos]
    return ciphertext
```

The above function works except in the case where the key is greater than the length of the `SYMBOLS` string. So, for example, if our `ciphertext` is "Z" and our key is 1. There is no character at position 26 in the `SYMBOLS` string (the last index is 25). So, we need to check for this case and "wrap around" to the beginning of the `SYMBOLS` string. How can we fix this? Well, suppose that our `ciphertext` is `Z` (index 25) and our key was 1, when we make `pos = pos + key`, the new char is at index 26 rather than 0. We could make it 0 by simply subtracting the length of `SYMBOLS` (which is 26). So, 25 + 1 = 26 - 26 = 0. Similarly, if our key was 2, then 25 + 2 = 27 - 26 = 1. And, of course, if our `pos` is not greater than the length of `SYMBOLS`, then we don't need to do anything. With the above in mind, we can add the corresponding letter to our ciphertext.

```
if pos >= len(SYMBOLS): #overflow when we add the key
    pos = pos - len(SYMBOLS) # 26 + 1 = 27; 27 - 26 = 1
else:
    ciphertext = ciphertext + SYMBOLS[pos]
```

Finally, we need to add the `else` statement if we get any characters that are not alphabetic. In this case, we simply add the character to the `ciphertext`. Here is the final function:

```
SYMBOLS = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'

def caesar_encrypt(plaintext, key):
    ciphertext = ""
    for char in plaintext:
        if char.isalpha():
            pos = SYMBOLS.find(char.upper())
            pos = pos + key # add the key val
            if pos >= len(SYMBOLS): #for wrap around
                pos = pos - len(SYMBOLS)
```

```
            ciphertext = ciphertext + SYMBOLS[pos]
        else:
            ciphertext = ciphertext + char
    return ciphertext
```

Let's test our function:

```
print(caesar_encrypt("ABC", 2)) # CDE
print(caesar_encrypt("ABC", 26)) # ABC
print(caesar_encrypt("XYZ", 1)) # YZA
```

With this basic Caesar encryption function in place, let's consider how to decrypt a message.

Caesar Cipher Decryption

The process of decrypting a message encrypted using the Caesar cipher is nearly identical to the process of encrypting a message. The only difference is that we subtract the key rather than add it. So, we can simply copy our `caesar_encrypt` function and change the line `pos = pos + key` to `pos = pos - key`. We'll call this function `caesar_decrypt` (notice that some of the variable names have been changed).

```
def caesar_decrypt(ciphertext, key):
    plaintext = ""
    for char in ciphertext:
        if char.isalpha():
            pos = SYMBOLS.find(char.upper())
            pos = pos - key # add the key val
            if pos >= len(SYMBOLS): #for wrap around
                pos = pos - len(SYMBOLS)
            plaintext = plaintext + SYMBOLS[pos]
        else:
            plaintext = plaintext + char
    return plaintext
```

Alternatively, we can simply pass in a negative key to our `caesar_encrypt` function. But, wait, suppose there is an "A" in our `ciphertext` and our negative key is -2. The `plaintext` then would be `Y`. An index of -2 is precisely what we want since a negative index counts from the end of the `SYMBOLS` string.

```
x1 = caesar_encrypt("ABC", 2) # CDE
x2 = caesar_encrypt("ABC", 26) # ABC
x3 = caesar_encrypt("XYZ", 1) #YZA
print(x1, x2, x3)
y1 = caesar_encrypt(x1, -5) # ABC
y2 = caesar_encrypt(x2, -26) # ABC
y3 = caesar_encrypt(x3, -1) # XYZ
print(y1, y2, y3)
```

Caesar Cipher Hacking

The Caesar cipher is very easy to hack. As mentioned, there are only 25 possible shifts, so one way to break the code is by brute force. A brute force method involves trying every possible combination. For example, suppose we have the cipher text "ITT LWOA IZM PIXXG." We can decrypt this message by considering all 25 possible shifts. Let's create a function that takes a string `encrypted_message` and returns a dictionary of all possible (1) keys (shifts) and (2) possible decryptions . We'll call this function `bf_caesar_hack` ("bf" for brute force).

```
SYMBOLS = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'

def bf_caesar_hack(encrypted_message):
    return decrypted_messages
```

We'll start by putting our encrypted message in uppercase. Then, we'll create an empty dictionary `decrypted_messages`. We'll construct the dictionary so its keys are its keys (the shifts) and the values of those keys are potential decrypted messages. Next, we'll loop through each key (0 to 25) and decrypt the message using our `caesar_decrypt` function. We'll call this message `pos_decryption` as it is a possible decryption. Finally, we'll construct the dictionary where the key is the key (shift) and the value is the `pos_decryption`. We'll return the dictionary.

```
def bf_caesar_hack(encrypted_message):
    encrypted_message = encrypted_message.upper()
    decrypted_messages = {}
    for key in range(0, len(SYMBOLS)):
        pos_decryption = caesar_decrypt(encrypted_message, key)
        decrypted_messages[key] = pos_decryption
    return decrypted_messages
```

Let's test our function. First, we started this section with the following ciphertext "ITT LWOA IZM PIXXG." To crack this code, we simply call our function and pass in the ciphertext.

```
SYMBOLS = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'

def caesar_decrypt(ciphertext, key):
    plaintext = ""
    for char in ciphertext:
        if char.isalpha():
            pos = SYMBOLS.find(char.upper())
            pos = pos - key # add the key val
            if pos >= len(SYMBOLS): #for wrap around
                pos = pos - len(SYMBOLS)
            plaintext = plaintext + SYMBOLS[pos]
```

```
        else:
            plaintext = plaintext + char
    return plaintext

def bf_caesar_hack(encrypted_message):
    encrypted_message = encrypted_message.upper()
    decrypted_messages = {}
    for key in range(0, len(SYMBOLS)):
        pos_decryption = caesar_decrypt(encrypted_message, key)
        decrypted_messages[key] = pos_decryption
    return decrypted_messages

print(bf_caesar_hack("ITT LWOA IZM PIXXG"))
```

Here are our results:

- Key 0: ITT LWOA IZM PIXXG
- Key 1: HSS KVNZ HYL OHWWF
- Key 2: GRR JUMY GXK NGVVE
- Key 3: FQQ ITLX FWJ MFUUD
- Key 4: EPP HSKW EVI LETTC
- Key 5: DOO GRJV DUH KDSSB
- Key 6: CNN FQIU CTG JCRRA
- Key 7: BMM EPHT BSF IBQQZ
- Key 8: ALL DOGS ARE HAPPY
- Key 9: ZKK CNFR ZQD GZOOX
- Key 10: YJJ BMEQ YPC FYNNW
- Key 11: XII ALDP XOB EXMMV
- Key 12: WHH ZKCO WNA DWLLU
- Key 13: VGG YJBN VMZ CVKKT
- Key 14: UFF XIAM ULY BUJJS
- Key 15: TEE WHZL TKX ATIIR
- Key 16: SDD VGYK SJW ZSHHQ
- Key 17: RCC UFXJ RIV YRGGP
- Key 18: QBB TEWI QHU XQFFO
- Key 19: PAA SDVH PGT WPEEN
- Key 20: OZZ RCUG OFS VODDM
- Key 21: NYY QBTF NER UNCCL
- Key 22: MXX PASE MDQ TMBBK
- Key 23: LWW OZRD LCP SLAAJ
- Key 24: KVV NYQC KBO RKZZI
- Key 25: JUU MXPB JAN QJYYH

Notice that the only key that produces a message that makes sense is key #8. So, we can conclude that the shift is 8 and the plaintext is "ALL DOGS ARE HAPPY."

Code

```
# Caesar Cipher Encryption, Decryption, and Brute Force Hack

SYMBOLS = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'

# Encryption with Caesar cipher
def caesar_encrypt(plaintext, key):
    ciphertext = ""
    for char in plaintext:
        if char.isalpha():
            pos = SYMBOLS.find(char.upper())
            pos = pos + key # add the key val
            if pos >= len(SYMBOLS): #for wrap around
                pos = pos - len(SYMBOLS)
            ciphertext = ciphertext + SYMBOLS[pos]
        else:
            ciphertext = ciphertext + char
    return ciphertext

# Decryption of Caesar ciphertext with key
def caesar_decrypt(ciphertext, key):
    plaintext = ""
    for char in ciphertext:
        if char.isalpha():
            pos = SYMBOLS.find(char.upper())
            pos = pos - key # add the key val
            if pos >= len(SYMBOLS): #for wrap around
                pos = pos - len(SYMBOLS)
            plaintext = plaintext + SYMBOLS[pos]
        else:
            plaintext = plaintext + char
    return plaintext

# Brute Force Hack of Caesar cipher

def bf_caesar_hack(encrypted_message):
    encrypted_message = encrypted_message.upper()
    decrypted_messages = {}
    for key in range(0, len(SYMBOLS)):
        pos_decryption = caesar_decrypt(encrypted_message, key)
        decrypted_messages[key] = pos_decryption
    return decrypted_messages

# Test 1 - ABC
plaintext1 = "ABC"
```

```
ciphertext1 = caesar_encrypt(plaintext1, 2)
print(ciphertext1) # CDE
plaintext1 = caesar_decrypt(ciphertext1, 2)
print(plaintext1) # ABC

# Test 2 - Hello World
plaintext2 = "Hello World"
ciphertext2 = caesar_encrypt(plaintext2, 4)
print(ciphertext2) # LIPPS ASVPH
plaintext2 = caesar_decrypt(ciphertext2, 4)
print(plaintext2) # HELLO WORLD

# Test 3 - All dogs are happy.
plaintext3 = "All dogs are happy."
ciphertext3 = caesar_encrypt(plaintext3, 7)
print(ciphertext3) # HSS KVNZ HYL OHWWF.
plaintext3 = caesar_decrypt(ciphertext3, 7)
print(plaintext3) # ALL DOGS ARE HAPPY.

# Test 4 - Brute Force Hack of "ITT LWOA IZM PIXXG"
print(bf_caesar_hack("ITT LWOA IZM PIXXG"))
```