Bacon's Cipher

David W. Agler

November 19, 2023

Bacon's Biliteral Cipher

Francis Bacon claimed that there are three virtues of ciphers: (1) they are easy to create and decipher, (2) it is impossible for any third-party to decipher, and (3) they don't alert suspicion (*Advancement of Learning*, Bk II, Sec. 6).

Bacon's cipher is a two-part cipher. The first part is a subsitution cipher, letters of plaintext are substituted with letters of ciphertext. The second part is a steganography cipher: the ciphertext is hidden in a message that doesn't look suspicious. The advantage of this approach is that we can hide the ciphertext in plain sight.

Substitution Cipher

As mentioned, the first part of Bacon's cipher is a substitution cipher. The substitution cipher is a cipher that replaces letters of plaintext with letters of ciphertext. In Bacon's case, each letter of plaintext is replaced by a group of five letters (a combination of "a" and "b").

Letter	Code	Letter	Code
A	aaaaa	N	abbab
B	aaaab	O	abbba
C	aaaba	P	abbbb
D	aaabb	Q	baaaa
E	aabaa	R	baaab
F	aabab	S	baaba
G	aabba	T	baabb
H	aabbb	U	babaa
I	abaaa	V	babab
J	abaab	W	babba
K	ababa	X	babbb
L	ababb	Y	bbaaa
M	abbaa	Z	bbaab

To encrypt the message "BACON", we would replace each letter with its corresponding code. That is, replace "B" with aaaab, "A" with aaaaa, "C" with aaaba, "O" with abbba, and "N" with abbab. The encrypted message would be aaaabaaaaaaaabaabbbaabbab.

Steganography Cipher

The next step is to employ a steganography cipher. A steganography cipher is a cipher that hides the ciphertext in a message that doesn't look suspicious. Let's call the text that we use to conceal the ciphertext the *concealment text*. For example, we might try to hide the message "Destroy the evidence" in the seemingly innocuous message "What time do you think breakfast is tomorrow? I hope they serve pancakes." In order to add the ciphertext to the concealment text, we need to find a way to represent the ciphertext using the letters of the message. One way to do this is to use two different typefaces or variations on the same typeface (e.g., bold and italic). In our example, we will use uppercase letters to represent "b" and lowercase letters to represent "a".

Let's illustrate this with the unencrypted (plain) concealment text "What time is breakfast tomorrow?". To conceal the ciphertext "aaaabaaaaaaaabaabbbaabbab" in this text, we'll go character by character through the ciphertext. Whenever the character is an "a", we'll use the lowercase version of the character in the concealment text. Whenever the character is a "b", we'll use the uppercase version of the character in the concealment text. We'll ignore spaces.

As our ciphertext "aaaabaaaaaaaabaabbbaabbab" begins with "aaaa", the first four letters of our concealment text will all be lowercase but since the fifth letter is "b", we'll make the fifth letter in the concealment text uppercase.

a	a	a	a	b	a	a	a
w	h	a	t	T	i	m	e

The result then is the following:

- Ciphertext: aaaabaaaaaaaabaabbbaabbab
- Plain concealment text: what tIme is breAkfAST tomorrow?
- Encrypted Concealment text: what tIme is breAkfAST tomorrow?

Now that we have our encrypted concealment text, we can send it to our recipient. If our recipient knows that the uppercase letters represents "b" and the lowercase letters represents "a". They will first identify the uppercase and lowercase letters in the concealment text. Then they will replace the uppercase letters with "b" and the lowercase letters with "a". In using this method, they can decrypt the concealment ciphertext to "aaaabaaaaaaaabaabbbaabbab". Finally, they can use the key to decrypt the ciphertext to "BACON".

Let's consider a final example of the Bacon cipher. It was only recently noticed on the tombstone of William and Elizebeth Friedman in the Arlington National Cemetery. Innocently enough, their grave contains the epigraph "Knowledge is Power". However, some of the characters were in serif and some were in sans-serif. Let's represent this by capitalizing the serif characters and leaving the sans-serif characters lowercase: "KnOwledGe Is pOwEr". If we replace the uppercase letters with "b" and the lowercase letters with "a", we get the ciphertext "babaa aabab aabab a". Decoding this text gives us "UFF". However, let's try to decode it using not the table considered earlier, but with the alphabet that Bacon used in his *The Advancement of Learning* (1605).

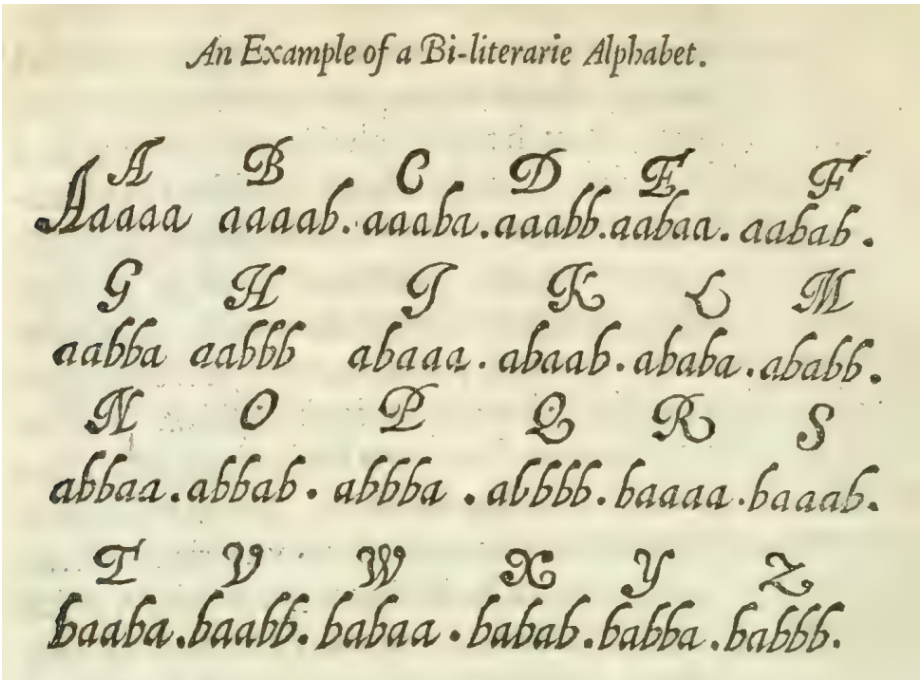

Figure 1: From *The Advancement of Learning*

In this alphabet, "I" and "J" share the same encoding, as do "U" and "V". Using this alphabet, "KnOwledGe Is pOwEr" is decoded as "babaa|aabab|aabab|a", which is decoded as "WFF", which are the initials of William F. Friedman".

Python Implementation

Here is a messy Python script that takes a plaintext message, turns it into A/B ciphertext, and then uses steganography to hide the ciphertext in a concealment text.

First, we will create a dictionary that maps each letter to its corresponding code.

```
bacon_key = {"A": "aaaaa", "B": "aaaab", "C": "aaaba", "D": "aaabb",
"E": "aabaa", "F": "aabab", "G": "aabba", "H": "aabbb",
"I": "abaaa", "J": "abaab", "K": "ababa", "L": "ababb",
"M": "abbaa", "N": "abbab", "O": "abbba", "P": "abbbb",
"Q": "baaaa", "R": "baaab", "S": "baaba", "T": "baabb",
"U": "babaa", "V": "babab", "W": "babba", "X": "babbb",
"Y": "bbaaa", "Z": "bbaab"}
```

Next, we will make use of a function to encrypt our plaintext. The function works by iterating through each character in the plaintext. If the character is a space, we will skip it. If the character is in our key, we will replace it with its corresponding code. If the character is not in our key, we will leave it as is. Thus, if there are any numbers or special characters in our plaintext, they will be left as is.

```
# Encrypt plain text as a string of a's and b's using bacon_key.
# Returns ciphertext.
def ab_encrypt(plain_text):
    ciphertext = ""
    for char in plain_text:
        if char == " ":
            continue # skip spaces
        if char.isalpha():
            char = char.upper()
            ciphertext += bacon_key[char]
        else:
            ciphertext += char
    return ciphertext
```

Next, we will implement the steganography to conceal the cipher text. To do this, let's create another function that takes the ciphertext and the concealment text as arguments. The function will iterate through each character in the concealment text. If the character is not an alpha, we'll add it to an empty string we'll call `encrypted_ct`. In contrast, if the character is an alpha, we'll check to see if we have reached the end of the ciphertext. If we have not reached the end of the ciphertext, we'll check to see if the character is an "a" or a "b". If the character is an "a", we'll use the lowercase version of the character in the concealment text. If the character is a "b", we'll use the uppercase version of the character in the concealment text. Finally, if there are any characters left in the concealment text, we leave them as is in the concealment text.

```
def make_unsus(ciphertext, concealment_text):
    encrypted_ct  = ""
    cip_index = 0
    for char in concealment_text:
        if not char.isalpha():
            encrypted_ct += char
            continue
```

```
        elif cip_index < len(ciphertext):
            if ciphertext[cip_index] == "a":
                encrypted_ct += char.lower()
            elif ciphertext[cip_index] == "b":
                encrypted_ct += char.upper()
            else:
                encrypted_ct += char
            cip_index += 1
        else:
            encrypted_ct += char
    return encrypted_ct
```

We can test our functions by encrypting a message and then hiding it in a concealment text.

```
plaintext = "The strike is tomorrow."
concealment_text = '''Dear John. I hope this letter finds you well.
Tomorrow is the first day of spring so I'm thinking of going for a
walk in the park or going for a bike ride.'''
ciphertext = ab_encrypt(plaintext)
encrypted_ct = make_unsus(ciphertext, concealment_text)
print(f"Plaintext message: {plaintext}")
print(f"Encryption in ciphertext: {ciphertext}")
print(f"Encryption using steganography: {encrypted_ct}")
```

Here is the result (I've added breaks and spaces to make it easier to read in LaTeX):

- Plaintext message: The strike is tomorrow.
- Encryption in ciphertext: baabbaabbbaabaabaababaabbbaaababaaaa babaaabaaabaaabaababaabbabbb aabbaaabbbabaaabbaaabababbbababba.
- Encryption using steganography: DeaR JohN. I HopE thIs lEtTer FINds yOu Well. tOmOrroW is tHe fiRst DaY of SPrING so I'M thiNKInG of gOIng fOr A WAlK iN The park or going for a bike ride.

Bacon decryption

Let's turn to decryption. There probably is a simpler way to do this that leverages the process of encryption, but I ended up writing the function from scratch. The function works by first converting the encrypted concealment text to a cipher. It checks if the text is alpha, and if it is, then it checks if it is uppercase (if it is, then "b") or lowercase (if it is, then "a").

```
def bacon_decrypt(encrypted_ct):
    def ct_to_cipher(encrypted_ct):
        ciphertext = ""
        for i in encrypted_ct:
```

```
            if i.isalpha():
                if i.islower():
                    ciphertext += "a"
                elif i.isupper():
                    ciphertext += "b"
            else:
                continue
        return ciphertext
    ciphertext = ct_to_cipher(encrypted_ct)
    print(ciphertext)
```

At this point, we have the ciphertext. Now we need to convert it back to plaintext using the Bacon key to map the ciphertext to plaintext. To do this, I simply swapped the keys and values in our key: the values become the keys and the keys become the values. Then I created two variables. One is `plaintext` to store the plaintext. Now I don't want to loop over each item in the ciphertext, but rather I want to loop over each group of five letters. So I created a variable `group` to store each group of five letters. I then looped over each letter in the ciphertext. If the letter is alpha, I added it to the group. If the group is five letters long, I added the corresponding plaintext letter to the plaintext variable and then reset the group variable to an empty string. If the letter is not alpha, I continued to the next letter. Finally, I returned the plaintext. This can't be the right way to do this, but it does work.

```
    def cipher_to_plain(ciphertext):
        swapped_key = {v: k for k, v in bacon_key.items()}
        plaintext = ""
        group = ""
        for i in ciphertext:
            if i.isalpha():
                group += i
                if len(group) == 5:
                    plaintext += swapped_key[group]
                    group = ""
                else:
                    continue
        return plaintext
    plaintext = cipher_to_plain(ciphertext)
    return plaintext
print(f'The plaintext: {bacon_decrypt(encrypted_ct)}')
```

Resources

1. The Bacon Cipher Explained by Cryptography for Everybody
2. PDF of Bacon's *On The Advancement of Learning*
3. CrypTool2
4. Article about the Friedman Tombstone

5. Any communication in and out of jail is closely scrutinized — so inmates turn to ingenious codes to convey secret messages. Business Insider, 2018.
6. Archived Page of FBI on Analysis of Criminal Codes and Ciphers. FBI, 2000.
7. Hidden Messages and Code Words: Bill Alldritt's Letters as a Prisoner in First World War Germany. By Robert Alldritt. Active History, 2016.

Code Used

```
# Bacon's Biliteral Cipher

bacon_key = {"A": "aaaaa", "B": "aaaab", "C": "aaaba",
"D": "aaabb", "E": "aabaa", "F": "aabab", "G": "aabba",
"H": "aabbb", "I": "abaaa", "J": "abaab", "K": "ababa",
"L": "ababb", "M": "abbaa", "N": "abbab", "O": "abbba",
"P": "abbbb", "Q": "baaaa", "R": "baaab", "S": "baaba",
"T": "baabb", "U": "babaa", "V": "babab", "W": "babba",
"X": "babbb", "Y": "bbaaa", "Z": "bbaab"}

# Encrypt plaintext
def ab_encrypt(plain_text):
    ciphertext = ""
    for char in plain_text:
        if char == " ":
            continue # skip spaces
        if char.isalpha():
            char = char.upper()
            ciphertext += bacon_key[char]
        else:
            ciphertext += char
    return ciphertext

# Steganography

def make_unsus(ciphertext, concealment_text):
    encrypted_ct  = ""
    cip_index = 0
    for char in concealment_text:
        if not char.isalpha():
            encrypted_ct += char
            continue
        elif cip_index < len(ciphertext):
            if ciphertext[cip_index] == "a":
                encrypted_ct += char.lower()
            elif ciphertext[cip_index] == "b":
                encrypted_ct += char.upper()
```

```
            else:
                encrypted_ct += char
            cip_index += 1
        else:
            encrypted_ct += char
    return encrypted_ct

plaintext = "The strike is tomorrow."
concealment_text = '''Dear John. I hope this letter finds you well.
Tomorrow is the first day of spring so I'm thinking of going for a
walk in the park or going for a bike ride.'''
ciphertext = ab_encrypt(plaintext)
encrypted_ct = make_unsus(ciphertext, concealment_text)

print(f"Plaintext message: {plaintext}")
print(f"Encryption in ciphertext: {ciphertext}")
print(f"Encryption using steganography: {encrypted_ct}")

# Bacon decryption
def bacon_decrypt(encrypted_ct):
    def ct_to_cipher(encrypted_ct):
        ciphertext = ""
        for i in encrypted_ct:
            if i.isalpha():
                if i.islower():
                    ciphertext += "a"
                elif i.isupper():
                    ciphertext += "b"
            else:
                continue
        return ciphertext
    ciphertext = ct_to_cipher(encrypted_ct)
    print(ciphertext)

    def cipher_to_plain(ciphertext):
        swapped_key = {v: k for k, v in bacon_key.items()}
        plaintext = ""
        group = ""
        for i in ciphertext:
            if i.isalpha():
                group += i
                if len(group) == 5:
                    plaintext += swapped_key[group]
                    group = ""
                else:
                    continue
```

```
            return plaintext
        plaintext = cipher_to_plain(ciphertext)
        return plaintext

print(f'The plaintext: {bacon_decrypt(encrypted_ct)}')
```