CSE350: Programming Assignment 1

Ananya Lohani (2019018), Mihir Chaturvedi (2019061)

Polyalphabetic substitution cipher (Vigenère Cipher)

In the Vigenère cipher, a secret keyword is used to encrypt a message by repeating it over and over to generate a keystream. This keystream is then used to encrypt the message by adding each letter of the message and the corresponding letter of the keystream modulo 26 (i.e., the number of letters in the alphabet). The result of this encryption is a ciphertext that can only be decrypted by someone who knows the secret keyword.

Encryption and decryption

Encryption

The encrypt function takes a plaintext input and a key, and returns the encrypted ciphertext by shifting each letter of the plaintext to the right by the corresponding letter in the key. The function works as follows:

1. Convert the key into a sequence of numerical values representing the ASCII codes of each letter in the key. The key is repeated as many times as needed to encrypt the entire plaintext.
2. For each letter in the plaintext, do the following:
    a. If the letter is a lowercase letter:
        i. Add the ASCII values of the current letter and the next letter from the key, modulo 26 (the number of letters in the alphabet).
        ii. Convert the result back to a character after adding the ASCII value of 'a'.
        iii. Add the result to the ciphertext string.
    b. If the letter is not a lowercase letter, simply add the letter to the ciphertext string without modification.
3. Return the ciphertext string as the encrypted ciphertext.

This implementation of the encryption function has a time complexity of **O(n)**, where n is the length of the plaintext, as it needs to iterate through each letter of the plaintext once.

Decryption

The decrypt function takes a ciphertext and a key as input and returns the original plaintext by undoing the encryption performed by the encrypt function. It works as follows:

1. Convert the key into a sequence of numerical values representing the ASCII codes of each letter in the key. The key is repeated as many times as needed to decrypt the entire ciphertext.
2. For each letter in the ciphertext, perform the following:
    a. If the letter is a lowercase letter, perform the following steps to decrypt it:
        i. Get the next numerical value from the sequence of ASCII values obtained in step 1.
        ii. Subtract the numerical value of the key letter from the numerical value of the ciphertext letter, modulo 26.
        iii. Convert the result back into a character after adding the ASCII value of 'a'.
        iv. Append the resulting character to the plaintext.
    b. If the letter is not a lowercase letter, append it to the plaintext without decryption.
3. Return the plaintext as the output.

The time complexity of this function is also **O(n)**, where n is the length of the ciphertext. This is because the function performs a constant amount of work for each letter in the ciphertext.

Hashing and verification

The hash function takes a plaintext input, and returns a hash value that is constructed from the input text. We use this hash function to construct plaintexts that are **recognizable**, i.e, those that satisfy the property: `p = (s, Hash(s))`.

The hash function works as follows:

1. Divide the input plaintext into blocks of N-character segments, where N is a constant specified in the implementation – in our implementation, N = 8.
2. For each block of characters, do the following:
    a. Take the first character of the block and initialize it as the initial hash value for that block.
    b. For each subsequent character in the block, perform the following:
        i. Add the ASCII values of the current hash value and the new character, modulo 26 (the number of letters in the alphabet).
        ii. Convert the result back to a character after adding the ASCII value of 'a'.
        iii. Assign this character as the new hash value for that block.
3. Concatenate all the hash values for each block to form the final hash value.

However, we do note that this hash function is simple and has several weaknesses, including the fact that it only uses a small portion of the input plaintext in constructing the hash, and it only uses lowercase letters of the alphabet in a predictable way.

Verification

We need the plaintext to be recognizable in order to verify whether a key correctly decrypts a given ciphertext in our brute force algorithm.

The verification function `is_recognizable` takes a plaintext input and determines whether it is a valid hash value. The function does the following:

1. Divide the plaintext input into two parts: the first part is used as the original plaintext, and the second part is the expected hash value.
2. Calculate the hash value of the original plaintext by calling the hashing function.
3. Compare the calculated hash value with the expected hash value.
4. If the two values are the same, return "True", indicating that the input is a valid hash value. If the two values are different, return "False", indicating that the input is not a valid hash value.

Brute-force solution

Since we know the key length to be 4, we can directly perform a brute-force attack by iterating through **all possible combinations of lowercase English alphabets of length 4**.

Total number of possible combinations = $\mathbf{26^4 = 4,56,976}$.
For a modern computer, iterating through all possible combinations should complete within a few minutes.

- At 100it/s: 1.27 hours
- At 1000it/s: 7.62 minutes
- At 4000it/s: 1.92 minutes [local benchmark]

The asymptotic time complexity of discovering the key via brute force is $\mathbf{O(26^k n)}$, where k is the length of the key and n is the length of the plaintext. This is because for each key, we call the decrypt function which has a time complexity of O(n).

Sample Inputs & Outputs

Constraints

- The plaintext used must be 567 (512 + 64 of its hash) characters long and consist only of lowercase alphabets [a-z] (empirically chosen via time taken for brute force).
- The key must be 4 characters long and consist only of lowercase alphabets [a-z] (given).
- The plaintext must be recognizable, i.e. of the form `p = (s, Hash(s))`

`KEY="zydf"`

1. **Input (plaintext):**
   `okacnqdvatlqphozmjwcboriefhffxblekmionhikpipruqfxnaqfxxweqsxeddjvlctljuk`
   `bomvyzqxdrhgutqsqevuguquhqeyqtalmqmqypcmqcklohfimmmzetfyhajceionoxcrfwpt`

bxkzjhvuhiaiahwtduifysbxvrjjqyzaqixchljizqwxrsnmmrhzlbprqichqonulveeavkq
fzsbeczdcbzfxxgtxrftemxardkfbuohcfytxwrfondbgdbxkcewndjjpdvwdorpoiimroyp
ofocnanbvbrsrqcyxwbwncwymmgoavkyblgmuqnmpbtktsmmzqsubsizjfhxerwhnpwzjrzn
eemukosdrynelbzoaqenhsjcejbakcibyhehnnjuexkijpurdxwkyrexbgmvgdvrwnewxxrc
xmqeeaarcxgajgadfqexdxsrxorqgzxuhqwujtqbobqjhkrvcravedtupvlpynyrkgnkctob
oqsnbhxbbdhjycvczgcttfvjfnmtyrgqbsjfazznmugikmzvncbqjhfrjtcwjwyxfogrikxp

Output (ciphertext):
nidhmogazrovofrelhzhamunddkkeveqdipnnlknjnluqstkwldvevabdovcdbgoujfykhxp
ampaxxtccpkltrtxpcyzfstzgohdprdqlopvxnfrpanqnfinlkpedridgymhdgrsnvfweusy
avneifyzggdnzfzycslkxqecupmopwcfpgahgjmnyozcqqqrlpkekzswpgfmpmqzkthjztnv
exvgdacibzckwvjywpiydkafqbnkasrmbdbywuuknlggfbecjahbmbmoobybcmuunglrqmbu
ndrhmyqguzuxqofdwuebmazdlkjtztndajjrtoqrozwpsqpryovzaqleidkcdpzmmnzeipcs
dcpzjmviqwqjkzctzohsgqmhdhefjalgxfhmmlmzdvnninxwcvzpxphcaepafbywvlhbwvuh
wktjdydwbvjfiedieohccvvwwmuvfxazgozzirtgnztogiuabpdadbwzotouxlbwjeqpbrrg
novsafagabkoxayhyefysdyoelpyxpjvaqmkzxcslsjnjkcamaevifiwirfbiubcemjwhiau

2. **Input (plaintext):**
antvglzszwfbfbyfgsrwyrksygxzvmaevovtpyalxuhzlnsmlbrofeandrtkbmxvifrztapi
eoqbcgvtymxfnoxisstsfnswnkkyeitdgblqtyckpmxtnxlhjkefevzwppbvupzhasfnjjsp
qvkaigxsasvevssxwqdzwgspxjawnfydrsoxjbtfewvzsyijbcsddxibskkngzwdfecpjexd
eyncxsistjebgbpxekryfowakmxhfbdqszhiswvmkuqtbtyhehydfnagjlqfqnqbrfqvkrid
eswawwucoabmzwsctrobthzitkxhwfckwuwwkcomgnfmhsqsnyxtzekebkokwjfdgxdgcbkr
frdukiiszgiimzdhefrpiqowpnlsbasyjktltmauxuuwmtuqxfgfynopkphtapbtgviwylkr
zjdaimlaozdlrgxhqdlcxeyczkuxaxktkxhzkqnmabopxqcejlqolvjwpwlccetgjmmgxxzv
tzchisxgjkclvznctfsbnltwpjytxvcbhdngaszbmkjtgqguurswqlqxwwwbipqchamxjdbe

Output (ciphertext):
zlwafjcxyuigezbkfqubxpnxxeaeukdjumyyowdqwskeklvrkzutecdscpwpakaahduesysn
dmtgbeyyxkakmmanrqwxelvbminddgwifzovswfpokaymvomiihkdtcboneatncmzqisihvu
ptnfheaxzqyjuqvcvogevevuwhdbmdbiqqrcizwkduyerwloaavicvlgrinsfxziecfuicai
dwqhwqlxshhgfzscdiudemzfjkamezgvrxknruyrjstyarbmdfbieldlijtkpltgqdtajpli
dqzfvuxhnyeryuvhsprgsfcnsiamvdfpvszbjarrflirgqtxmwayycnjairpvhiifvglbznw
epgzjglxyelnlxgmdduuhorboloxayvdiiwqskdzwsxblrxvwdjkxlrujnkyzneyftlbxjnw
yhgfhkofnxgqqeampbohwcbhyixczvnyjvkejoqrzzruwofjijttktmbouohbcwlikplwvca
sxfmhqaliifquxqhsdvgmjwbohbywtfggbqlzqcglimyfojztpvbpjtcvuzghnthgypcibej

3. **Input (plaintext):**
fpotpsriyyfwknucqbaikfrimtvebypnueslrpmlqydzpjqrdmiapurvnshfxzieuieqztih
falglksgbscjxsvnjuorpehcycokrdaytmfyugqdsfjhziefymbwlnzrfrfxgrabtvcdlffb
uhusdzpnmqubmxbzildifhlklhwdtmhloxqafrwybjtrfeomrfparfjunswgxpwzthwzcnxm
gfawtfctufqsosfqbzorayktkqkpycizpiatnobnygcfqetulaoqkvgjizcsfyjpipmsqozx
jxlsztfqunoaetpkvjxqqyxtswwglthcyxcksdzzwtfvpnrwjkejsbyhdezpjuefvmpncywn

jcnhusnvlkfadkrgbvbbxhiejppbglnpnechmhawxbqevzzoafykniscdbrwlnletidskafb
aoanlkofgacnkpekjtzlpyhfwsivkxhjbdhilxzlpflbogywywgiyotbxipqbpzouhpdyvzp
nohbhnzrhqnfevszdpbkqbdvwprglordkotaiggfsjcbwrvdaeehszkohpzcempiloluonat

Output (ciphertext):

Enryoqunxwibjlxhpzdnjdunlryjawsstcvqqnpqpwgeohtwcklfosuamqkkwxljtghvyrlm
eyolkivlaqfowqysisrwockhxarpqbddskidtetirdmmyghkxkebklcwepicfpdgstfikdig
tfxxcxssloxglveehjgnefopkfziskkqnvtfepzdahwwecrrqdsfqdmzmqzlwnzesfzeblar
fddbsdfytdtxnqivaxrwznyjonuxaleogdymmesxefkpcwzkyrvjtjohxfxewmuhnpxpmcc
ivoxyrivtlrfdrspuhavpwayruzlkrkhxvfprbcevriaolubiihorzbmcccuishkukssbwzs
iaqmtqqakiifciulategwfljinsgfjqumcfmlfdbwztjuxctzdbpmgvhczubklojsggxjyig
zmdskirkfyfsjnhpircqowkkvqlajvkoabknkvcqodognebbxujnxmwgwgsvancttfsixtcu
mmkgglcwgoqkdtvecneppzgavnulkmuijmwfhejkrhfgvpyizchmrxntgnchdksnkmoznldy

4. **Input (plaintext):**

cjwdnjpmwipqxkrkdxhlumahrmvqrnumspagkvpfmhkyneenlockzqxxuegbehogiemraoas
kuvvunisikyqmqwourlapfwoemliktogyqkgaphhvvjtqbnobenzdrhhhgjjilugpvrzfkqu
wfngrathzyvdkckhvxrjgtruhunnkxwovjvoqrgmkmejxbghvhlbhmjrpjchcdjuvtovoeru
qhtbuzzihvxdxntygyxxhjsuynekeregwvltsotdbuigcqwcvsyuwespxkmzfacoktmrnfyn
lqgkzgeindbhhdkhtfwxsxgbpcmdazhwxktuiwvccixxndlsrdijbsjpsdwkfgansuofezzf
jooridnmuuwvlwsgiombhsnfugsqycyglvnyqmllgobhtlvujcnvljsecujtsmlqvlqeeucw
yyjtbqojpwbuewztacyzqrvmgpmuoqlkybfcxxoxpuvtyretmcafmzjcouzcvhorympjlirn
jvabwjqghrfymjukvbsaghkzyzlycsmuhpardaexzmnjiznivxczmmkampvjdwmynaljpqfg

Output (ciphertext):

bhzimhsrvgsvwiupcvkqtkdmqkyvqlxrrndljtsklfndmchskmfpyoactcjgdfrlhcpwzmdx
jsyatllxhibvlozttpofodztdkonjrrlxonlznkmutmypzqtacqecpkmgemohjxlotueeitz
vdqlqywmywyijanmuvuofruzgsqsjvztuhytppjrjkhowzjmufoggkmwohfmbbmzurrancuz
pfwgtxcngtaiwlwdfwacghvzxlhpdphlvtoyrmwiasllbozhuqbzvcvuwipeeyftjrpwmdbs
kojpyehnmbemgbnmsdzcrvjgoapizxkbwiwzhuyhbgacmboxqbloaqmurbzpeedsrsrkdxck
imrwhbqrtszakuvlhmpggqqktevvxablktqdpkoqfmemsjyziaqakhvjbsmyrkovujtjdsfb
xwmyaoroouezducyzabeppyrfnpznoopxzihwvrcosyyxphyladklxmhnschufrwxksokgus
itdgvhtlgpidlhxpuzvfffnexxodbqpzgndwcyhcykqohxqnuvfelknflnyocupdmyooooil

5. **Input (plaintext):**

bqloyflbhyhqqtswyhhbqnbawicplnpskdhgjlthzmvwrjmtutjfgbncjzhckjuuwmqtkrzy
xxirvggvbecwsibkfpdasjvtukaqszatwfhckvbboxmorxddxhybgphbxgyfslihhwshgjle
sdqrlwfnercmxlhuyhcugciiwfdwyvymajhojbzdtfrofisahcgjnhcbxgdiwgcaiouradkl
ubdorbptgiilullkgcsclzighncxlnorxgpywdksrltytmyuakapfdkyceoskgvooexmvpzl
rkrqilimxtalkjqgukjoqylyugdasfktvpxuxliksrgshlfobicrruuptuvomsglmqbgpyhz
gqiutubcanghjgxlbqmgeukierjdspoqbmzkvllufhejvmibwhsilafvvtdaskjknorbeexd
gaunqcaqumktnexrpujuylhrrdzkikgxpgzvjwzuxpsbcpwoyqcrvjqbsbfxojhibyjdlkrp
wqsyhcngfzrauhxypvomerfgygbszdqivsfmhawrqplvvqydbswrcoxzshpombvotyngchme

Output (ciphertext):

```
aootxdoggwkvprvbxfkgplefvgfuklsxjbklijwmykybqhpytrmkfzqhixkhjhxzvktyjpcd
wvlwuejaacfbrgepengfrhyytidvrxdyvdkhjtegnvptqvgiwfbgfnkgwebkrjlmguvmfhoj
rbtwkuisdpfrwjkzxffzfalnvdgbxtbrzhktizcisdutegvfgajomffgwegnveffhmxwzbnq
tzgtqzsyfglqtjopfavhkxllglfcklrwwesdvbnxqjwdskbzziduebndbcrxjeytncaruncq
qiuvhjlrwrdqjhtltimtpwodtegfrdnyunazwjlprpjxgjitagfwqsxussytlqjqloelowke
folzssehzljmieaqaopldsnndpmirnrvakcpujozefhouklgvfvnkyiaurgfrimpmmugdcai
fyxspadvtknymcawosmzxjkwqbcphijcoecaiuczwnvgbnztxofwuhtgrzicnhknawmikiuu
vovdgaqlexuftfadotrrdpilxeexybtnuqirgyzwpnoauobiaqzwbmaerfstlzytswqlbfpj
```