Assignment 1 - Project 0 : Mono Alphabetic Substitution Cipher

Aman Mehra (2017017) **Aditya Bhadoo(2017008)**

Assumptions -

1. The universe of alphabets in the plaintext is **U** = {**aa,ab,ac,ba,bc.bb.ca,cb,cc**}
2. The set of plaintexts encrypted with a certain key **k** have at least one occurrence of every alphabet in **U**. This is necessary to ensure that the bruteforce attack terminates with a unique key rather than a set of possible keys.

Encryption-Decryption System -

The entire codebase is built in C++. To generate a key, a random permutation of the alphabets in **U** is mapped to a fixed ordering of the alphabets - {**aa,ab,ac,ba,bc.bb.ca,cb,cc**}. Two maps (**F** and **B**) are created to represent both the forward (**F**) and backward (**B**) mapping of the two orderings.

The forward mapping (**F**) is used to look up the corresponding ciphertext for a given plaintext alphabet. Thus this mapping is used to perform encryption on a given stream of plaintext.

The backward mapping (**B**) is similarly used to look up the corresponding plaintext for a given ciphertext alphabet. Thus this mapping is used to perform decryption on a given stream of ciphertext.

To ensure correctness of the decrypted ciphertext, a hash of the plaintext is appended to the generated ciphertext. In practice we also append the length of the hash at the end for ease of processing. Thus for a given plaintext (**p**), hash function (**h**), encryption function (**F**) with key **k**, the generated ciphertext **c** is -

$$\mathbf{c = F(p,k) \,||\, h(p) \,||\, length(h(p))} \quad \textbf{(1)}$$

Here a || b refers to appending strings a and b.

At the time of decryption, the function represented by **B** is applied to retrieve the decrypted plaintext $\mathbf{p_d}$. To verify if the decrypted plaintext actually is correct the hash is computed and compared with the hash of the original plaintext, i.e.,

$\mathbf{p_d = B(c,k)}$ and $\mathbf{h(p_d) == h(p)}$. **(2)**

Attack -

We run a brute force attack on the ciphertext by iterating over all possible permutations (**9!**) of the alphabets in **U.** All keys which produce decrypted plaintexts having their hash matching the hash of the plaintext (Eqn **2**) are added to the set of possible keys.

Iterating over all the ciphertext produced by a particular key, we eliminate candidate keys that do not work on all the ciphertexts. This way we bring down the number of candidate keys to the minimal number possible given the plaintext. Using assumption **2**, this number is **1.** Thus we end up with a unique key for decrypting the plaintext.

Code -

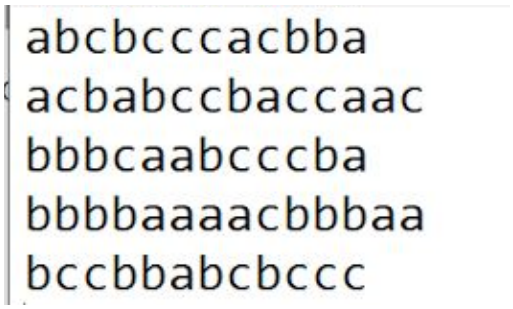
The code is arranged in two files - encryption_decryption.cpp and bruteForceAttack.cpp. Executable files for the same have been created for both as enc_dec.out and attack.out respectively.

enc_dec.out - Generates the key and performs encryption using this key on all plaintexts in file input.txt. Subsequently decryption is performed according to eqn **2**, to verify the correctness of the approach. The ciphertexts are output to the file cipher.txt.

attack.out - Runs a bruteforce attack on all the ciphertexts and outputs the set of feasible keys. When assumption **2** holds, a unique key is output and it can be compared with the key produced by enc_dec.out to verify correctness. The decrypted plaintext is stored in output.txt.

Demo -

Plaintext to encrypt -

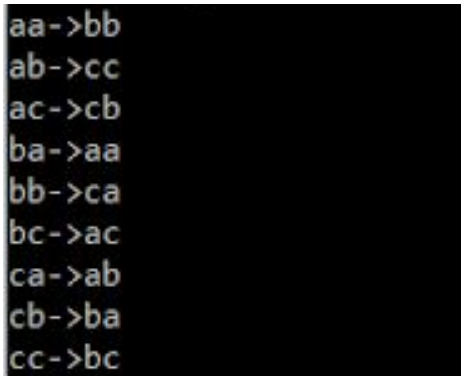


Generated Key -

Ciphertext -

```
ccbabcabbaaa33805679174811565418
cbaaacbacbabcb1268001837377318549320
caacbbacbcaa929647969528702121819
cacabbbbbacabb981125482148766023619
acbaaaacacbc1120695004701069834520
```

Key Generated by Attack -



Plaintext decrypted by attack -

```
abcbcccacbba
acbabccbaccaac
bbbcaabcccba
bbbbaaaacbbbaa
bccbbabcbccc
```