# CMPUT333 – Assignment 1 – Report

# Part 1

The decrypted plaintext is as follows:

A Polar Explorer

All the huskies are eaten. There is no space

left in the diary, And the beads of quick

words scatter over his spouse's sepia-shaded face

adding the date in question like a mole to her lovely cheek.

Next, the snapshot of his sister. He doesn't spare his kin:

what's been reached is the highest possible latitude!

And, like the silk stocking of a burlesque half-nude

queen, it climbs up his thigh: gangrene.

Joseph Brodsky

The key used to encrypt the plaintext is “2brodsky”. The key was determined in two stages. The first stage was determining the key length. In order to find the key length, the following procedure was used. We took the ciphertext and began by shifting it right one character. Then we compared the un-shifted ciphertext to the shifted ciphertext, and counted how many characters matched between the two. We then incremented the amount by which the ciphertext is shifted, and kept track of how many matches occurred each time. The shift amount that resulted in the highest amount of matches would indicate the length of the key.

The reason why this technique works is because each character of the key will encode plaintextLength/keyLength amount of characters. In that number of plaintext characters, there is a chance that characters will repeat, and be encoded by the same key character, to the same ciphertext character. When you offset the ciphertext by the same amount of spaces as the length of the key, those characters that were the same will match up, thus resulting in a higher amount of matching characters than offsetting by an arbitrary amount.

The process of finding the key length was mostly automated. We looked at the number of matches in offset amounts from 0 to 20, and the amounts with the highest number of matches were 8 and 16. The key length therefore must have been the smaller one of the two.

In order to find the key itself, we assumed that the plaintext can only consist of readable ASCII characters. Using this assumption and the fact that the key could only consist of alphanumeric characters, we could narrow down the list of possible characters for each character of the key. For each key character, we began by picking a character from a list of the possible alphanumeric characters, and decoding each ciphertext character that matches that key character (character of position i, i+keyLength, i+2keyLength, etc.). The idea was that if any of the plaintextLength/keyLength ciphertext characters decoded to a non-readable ASCII character, we would discard that key character and move on to the next possible alphanumeric key character. We ended up with some key characters where more than one key character would decode all its ciphertext characters to readable ASCII, and other cases where there were no key characters that decoded all ciphertext characters to readable ASCII. This process ended up being semi-automated. We looked at which key characters decoded majority of their ciphertext characters to readable ASCII, and used these possible characters to construct 3 possible keys by permuting the available characters. This is the output we produced in our program that helped us construct the key:

======TESTING FOR position 0

matched 56 for key 2

======TESTING FOR position 1

matched 56 for key b

======TESTING FOR position 2

matched 56 for key r

======TESTING FOR position 3

matched 56 for key o

======TESTING FOR position 4

matched 55 for key d

======TESTING FOR position 5

matched 55 for key c

matched 55 for key s

matched 55 for key S

======TESTING FOR position 6

matched 55 for key k

======TESTING FOR position 7

matched 56 for key y

The way we interpreted the output is that for all the key characters except for char 5, there was only one character out of the possible alphanumeric characters that resulted in 55 or 56 readable ASCII characters. We used the 3 possible characters in position 5 to come up with the 3 possible keys : 2brodcky, 2brodsky, and 2brodSky.

We then decrypted the ciphertext with each of the keys and looked for readable English text. ‘2brodsky’ was the key that resulted in all readable plaintext characters.

If the plaintext was compressed before it was encrypted, our strategy would have to be different. Each compression tool has its own algorithm, so in order to be able to find a decryption strategy, we would need to understand how the compression tool archives the plaintext. For the command line tar tool, it simply creates a padding and meta-data before and after the plaintext, leaving the plaintext intact, which wouldn’t affect our decrypting strategy much as long as we account for the characters used in the padding. Other tools such as ones that produce .zip files work by analyzing the text and creating a dictionary of words or phrases that repeat, and the next time that phrase repeated, it would just reference its location in the dictionary. Looking at the binaries produced by these files, it is all still mostly readable ASCII used to reference the dictionary, with a few unusual characters. We could still use the same technique and look at key characters that produce the highest amount of readable ASCII characters.

# Part 2

The decrypted plaintext is as follows: