Group P

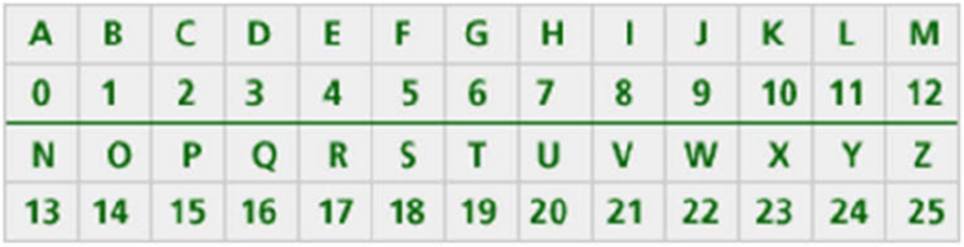
[redacted]

Dear OCRAI,

Thank you for your letter. We realize that, especially in these times of competition and of computing progression, the need for secure cryptosystems becomes increasingly important and needed. We commend your ability in identifying the vulnerabilities within your process. Mobile phone usage has long been a security vulnerability, and we hope that our proposed cryptosystem will satisfy your needs as a company.

In order to properly encrypt and secure the data, we have endeavored to find a cryptosystem and a corresponding algorithm in order to code the messages, per the company's request. The method we describe here has been constructed to be a combination of the Vigenère cipher and the Hill Cipher. We hope that the duality of this cipher will help prevent attempts to decode the messages by those who are not meant to read them, and we will outline the reasons why we believe it will be so.

To encrypt using our process, we must first obtain the first part of a key which is 4 to 6 letters long. We convert these letters to numbers (by assigning a=0, b=1, and so on until z=25). This algorithm will then roll through each letter of the message (omitting spaces) and shift the letter by that number of letters in the alphabet. An alternative way of calculating this, is assigning each letter of the plaintext to a number (using the system just discussed) and add the key's number to that letter. We use the following table in our example.



A	B	C	D	E	F	G	H	I	J	K	L	M
0	1	2	3	4	5	6	7	8	9	10	11	12
N	O	P	Q	R	S	T	U	V	W	X	Y	Z
13	14	15	16	17	18	19	20	21	22	23	24	25

For example, if the key is "OCRAI", then the corresponding shift factor is [14, 2, 17, 0, 8]. Let us have a plaintext that begins with "Four score and seven…" We now convert this sequence to their corresponding numbers, which gives us

5 14 20 17 18 2 14 17 4 0 13 3 18 4 21 4 13

We then add each key to each letter, adding the first letter of the key to the first letter of the plaintext, second to second, and so on, and once we reach the end of the key (in this case, after 5 letters), we begin again. We note that if our resulting number is greater than or equal to 26, we subtract 26 from it. We also note that if the message has an odd number of letters, we will add a random letter to the ciphertext at the end of the message, which is necessary for the next step in our cipher. So our resulting ciphertext will be based on (where R is a random number chosen to make the number of letters even).

19 16 1 17 0 16 16 8 4 8 1 5 9 4 3 18 15 R

We note that multiple times in our cipher, we had to subtract 26 in order to keep the answer between 0 and 25.

We now explain the second part of the 2-step cipher. We now manipulate the cipher using Linear Algebra to further encrypt our data. The second part of the key should be 4 letters long, which will correspond to 4 numbers. In our case, let our full key be OCRAIFIRD. These last 4 numbers will represent a matrix, which will look like

$$\begin{matrix} c & d \\ e & f \end{matrix} = \begin{matrix} 5 & 8 \\ 17 & 3 \end{matrix}$$

We then take numbers from our previous ciphertext, two at a time, and multiply them by the matrix. So, given letters a and b, and numbers in our 2 x 2 matrix c, d, e, f, we can multiply them, and the resulting formula is

$$a*c+b*e \qquad \text{and} \qquad a*d+b*f$$

two numbers which will replace what we previously had in our ciphertext. Using our previous example and key, this will give us (using S in computations using R):

3 18 8 7 12 22 8 22 0 4 12 23 9 6 9 0 S S

Giving us the resulting ciphertext

dsgjmwiwaemxjgjaSS

We note that in our choice of a key, the first 4-6 letters are completely arbitrary, but the last 4 (for the matrix) must be a matrix that is invertible (modulus 26). The way that we can ensure that a matrix is invertible (mod 26) is seeing if gcd(26,(cf-de))=1.

In order to decode, we must find the invertible matrix of the key, and follow the same process in reverse (multiplying two elements at a time to the inverted matrix and subtracting the first key amount).

We have performed a frequency analysis, and have discovered that in this process, the frequency of the letters is very similar, eliminating most possibility to decrypt using that method. Eliminating spaces also eliminates the ability to use word length to determine the message. The double layer of encryption adds an extra layer of security, making it difficult to test all of the combinations of the two keys. The Vigenère cipher has the weakness of being able to test all the combinations of cipher lengths, and from that be able to determine the key. The duality of this cipher covers that weakness. The Hill cipher alone has the vulnerability of a plaintext attack, and by having a piece of plaintext and the corresponding cipher, the matrix key is easily obtained. Again, this duality covers this vulnerability.

For your convenience, we have included a Python file that automates the process of encryption and decryption.

Sincerely,

[redacted]

Independent Mathematical Contractors, Inc.

```
#super_cipher.py
import sys
import re
import numpy as np

def apply_shift_cipher(digittext, key, shift_len):
    for i in range(len(digittext)):
        alpha = key[i % shift_len]
        digittext[i] = (digittext[i] + alpha) % 26
    return digittext

def multiply_pairs(digittext, matrix):
    output = []
    for i in range(len(digittext) // 2):
        pair = np.array([digittext[2*i], digittext[2*i+1]])
        output_pair = np.matmul(pair, mat) % 26
        output.extend(output_pair.tolist())
    return output

def digits_to_string(output_in_digits):
    output = ""
    for d in output_in_digits:
        output += chr(d + 97)
    return output

args = sys.argv
if len(args) != 5:
    print("Usage: python super-cipher.py <encode/decode> <key> <input-filepath> <output-filepath>")
    exit()
key = args[2]
# Note, the first part of the key can be the same for decoding and encoding, but the last
# 4 digits must be the inverse matrix mod 26 for decoding
in_filepath = args[3]
out_filepath = args[4]

input_text = ""
with open(in_filepath) as f:
    input_text = f.read()
# Only keep letters, and make everything lowercase
input_text = re.sub(r"[^a-zA-Z]", "", input_text).lower()

# Turn letters into list of digits 0-26
digittext = [ord(c) - 97 for c in input_text]

# Make key and matrix nice formats
key = [ord(c) - 97 for c in key]
shift_len = len(key) - 4
mat = np.array([[key[shift_len], key[shift_len + 1]], [key[shift_len + 2], key[shift_len + 3]]])
```

```
# Pad message if odd number of characters
if len(digittext) % 2 == 1:
    digittext += [np.random.randint(0, 26)]

if args[1] == "encode":
    shift_output = apply_shift_cipher(digittext, key, shift_len)
    output_in_digits = multiply_pairs(shift_output, mat)
else:
    matmul_output = multiply_pairs(digittext, mat)
    output_in_digits = apply_shift_cipher(matmul_output, [-k for k in key], shift_len)
output = digits_to_string(output_in_digits)
print(output)
with open(out_filepath, "w+") as f:
    f.write(output)

#char_freq = {}
#for c in output:
#    char_freq[c] = char_freq.get(c, 0) + 1
#print(char_freq)
```