

Question 1

The Extended Euclidian Algorithm

$$\begin{array}{lll}
 r_0 = a & s_0 = 1 & t_0 = 0 \\
 r_1 = b & s_1 = 0 & t_1 = 1 \\
 r_{m+1} = r_{m-1} - q_m r_m & s_{m+1} = s_{m-1} - q_m s_m & t_{m+1} = t_{m-1} - q_m t_m \\
 & q_m = \left\lfloor \frac{r_{m-1}}{r_m} \right\rfloor &
 \end{array}$$

Theorem

$$r_m = r_0 s_m + r_1 t_m$$

Proof

Base Case ($m = 0$)

$$\begin{aligned}
 r_0 &= r_0 s_0 + r_1 t_0 \\
 &= r_0 1 + r_1 0 && \text{Substitute for } s_0 \text{ and } t_0 \\
 r_0 &= r_0
 \end{aligned}$$

Base Case ($m = 1$)

$$\begin{aligned}
 r_1 &= r_0 s_1 + r_1 t_1 \\
 &= r_0 0 + r_1 1 && \text{Substitute for } s_1 \text{ and } t_1 \\
 r_1 &= r_1
 \end{aligned}$$

Inductive Step. Assuming $r_m = r_0 s_m + r_1 t_m$, we will prove that $r_{m+1} = r_0 s_{m+1} + r_1 t_{m+1}$ for any $m > 0$.

$$\begin{aligned}
 r_{m+1} &= r_0 s_{m+1} + r_1 t_{m+1} && \text{To Prove} \\
 &= r_0 (s_{m-1} - q_m s_m) + r_1 (t_{m-1} - q_m t_m) && \text{Substitute for } s_{m+1} \text{ and } t_{m+1} \\
 &= r_0 s_{m-1} - r_0 q_m s_m + r_1 t_{m-1} - r_1 q_m t_m && \text{Multiply by } r_0 \text{ and } r_1 \\
 &= r_0 s_{m-1} + r_1 t_{m-1} - r_0 q_m s_m - r_1 q_m t_m && \text{Rearrange the terms} \\
 &= r_{m-1} - r_0 q_m s_m - r_1 q_m t_m && \text{Substitute } r_{m-1} \text{ by the inductive hypothesis} \\
 &= r_{m-1} - q_m (r_0 s_m + r_1 t_m) && \text{Factor out } q_m \\
 &= r_{m-1} - q_m r_m && \text{Substitute } r_m \text{ by the inductive hypothesis} \\
 r_{m+1} &= r_{m+1} && \text{Substitute } r_{m+1} \text{ by the definition of } r_{m+1}
 \end{aligned}$$

Question 2

I used a program I wrote (SimpleCiphers, attached) to find the matrix for the key that encrypts the plaintext, `breathtaking`, to the ciphertext, `RUPOTENTOIFV`. After putting the plaintext and ciphertext in the files `hill-pt.txt` and `hill-ct.txt`, I ran the following command:

```
b@transam:~/work/ritcrypto$ ./SimpleCiphers hill pbreak eng hill-pt.txt
hill-ct.txt
Possible decryption #1, key: 3,21,20/4,15,23/6,14,5
breathtaking
```

This indicates that the matrix:

$$\begin{bmatrix} 3 & 21 & 20 \\ 4 & 15 & 23 \\ 6 & 14 & 5 \end{bmatrix}$$

translates the given ciphertext to `breathtaking`.

The method used to find the key uses the following properties of the plaintext, the ciphertext, and the key.

- $P \times K = C$ - Any given plaintext matrix multiplied by the key gets the corresponding ciphertext matrix.
- $P^{-1} \times P \times K = P^{-1} \times C$ - We can multiply both sides by P^{-1} and not change anything.
- $K = P^{-1} \times C$ - Therefore, the key is the the inverse of a plaintext matrix times the corresponding ciphertext matrix.

The first thing the program does is guess a possible size for the matrix ¹. Next the plaintext and ciphertext are broken up in to chunks that are the same size as the width of the key. Next all possible n element permutations of these plaintext/ciphertext vectors are calculated ² (where n is the height of the matrix).

Now we have a list of possible values for C and P in the equation above. Next the program finds the first permutation whose plaintext component is invertable. Once an invertable plaintext matrix is found finding the key for the guessed size is simply a matter of multiplying the ciphertext matrix by the inverted plaintext matrix. Finally, one last check is done to see if the calculated key matrix is invertable. (If it isn't it can't be the key).

The result of the above calculations is a list of 0 or 1 key matrices for each possible key size. Finding the correct key is simply a matter of finding the key that produces the plaintext that is obviously correct.

Question 3

Part 1

If a cipher has diffusion than changing one character of the plaintext should change several characters of the ciphertext and changing one character of the ciphertext should change several charac-

¹Actually it iterates over all possible key sizes and picks the best one

²This is done lazily so permutations are only generated until a satisfactory one is found

ters of the plaintext. Most block ciphers have this property. The cipher described in this question **does not** have this property. Like the Vigenère cipher, this cipher encrypts characters one at a time.

Part 2

If a cipher has confusion then each character of plaintext or ciphertext should depend on many different parts of the key. An example of a cipher with confusion is the Hill cipher. Changing each character depends on an entire column of the key. This cipher **does not** have this property. Like the Vigenère cipher, each encrypted character depends only on a single element of the key.

Part 3

Modifying the Kasiski test to determine the key length for this cipher would not work. An unmodified Kasiski test wouldn't get you anywhere because of the additional shifts after each cycle of the key. The only way you'd be able to make the Kasiski test work is to know the length of the key and compensate for the additional shifts, but if you already know the key length... then there is no point in running the Kasiski test.

Part 4

The index of coincidence method, however, can be modified to work on this cipher. To find the key length of the Vigenère cipher using the index of coincidence method the ciphertext is shifted to the left by n and compared with the original ciphertext. The value of n that yields the most matching characters between the two ciphers is the most probable key length. This works because certain letters show up more frequently than others. Once you shift by the same amount as the size of the key "Es" start matching up with other "Es", etc.

Modifying this method to work for this cipher (the one described in the homework question) requires making the original cipher text "look" like ciphertext encrypted with the Vigenère cipher. Because we're already guessing a key length when we use the index of coincidence method (unlike the Kasiski test) we can compensate for the additional shifts by subtracting n everywhere n was added by the encryption function. If we're guessing the key length is 4 then we subtract 1 from elements 4, 5, 6, and 7 of the ciphertext, subtract 2 from 8, 9, 10, and 11, etc. This reverses the additional additions caused by the cycling of the original key. After these have been reversed we can treat the ciphertext as if it was ciphertext encrypted with the Vigenère cipher to find the key length.

Finding the actual keyword follows the same process as finding the key length. First we compensate for the additional shifts, then use the same process as we do for the Vigenère cipher. To find the key for the Vigenère cipher (once the key length is known) we first break the ciphertext up into n groups where each group contains every n th element of the ciphertext (where n is the key length). So if n was 4, the first group would contain elements 0, 4, 8, ... the second 1, 5, 9, ... etc. Then the frequency distribution of the letters in each group is compared to the frequency distribution of the assumed language shifted by various amounts. If a given group matches up to the frequency distribution of the language shifted by x then x is probably the shift. This process is repeated for each group until the entire key is found.

Part 5

I used a program I wrote (SimpleCiphers, attached) to break the given ciphertext. After putting the ciphertext in hw2-cipher.txt I ran the following command:

```
b@transam:~/work/ritcrypto$ ./SimpleCiphers hw2 br eng hw2-cipher.txt
Possible decryption #1, key: duck
theproblemofdistinguishingprimenumbersfromcompositenumbersandofresolvi
ngthelatterintotheirprimefactorsisknowntobeoneofthemostimportantanduse
fulinarithmeticithasengagedtheindustryandwisdomofancientandmoderngeome
terstosuchanextentthatitwouldbesuperfluoustodiscusstheproblematlengthf
urthertedignityofthescienceitselfseemstorequirethateverypossiblemeans
beexploredforthesolutionofaproblemsoelegantandsocelibratedkarlfriedric
hgaussdisquisitionesarithmeticaeeighteenhundredandone
```

This indicates that the key “duck” results in the plaintext shown. SimpleCiphers uses the method described above to break the ciphertext.