Group 2

3/4/20

Lab 4 Report

    In this lab we will be encrypting a string message through an affine cipher and then rsa encryption. After, we will decrypt that encrypted message through rsa decryption and then affine decryption to produce the original string message.

For affine encryption, the values we used were **a = 3, b=4**, producing the formula **f(p) = (3p+4)mod26.** The first step was to translate the letters in the string to their respective numerical value. To do so, we incorporated a dictionary with the letters as they key (A-Z) and number as the value (0-25). From there we can append those translated numbers into a list and then perform the affine encryption to each number in the list. Ex: **(a\*list[i] + b)%26.** The final step is to translate the numbers into letters through the trans\_eng function. As a result, we have a string message that has been affine encrypted into a list.

After performing the affine encryption, we can pass that resulting list into the rsa\_encrypt function. First we must translate the letter encrypted list into numbers, which can be done through calling the dictionary keys. For rsa, our key values were **p = 53, q = 61, e = 11**. Before using the keys, we had to test that the greatest common factor of (p-1)(q-1) \* e was 1. The gcf of 3120 and 11 is 1. Since **53\*61 is 3233**, we encrypt in blocks of 4 because **2525 < 3233 < 252525.** To create the 4 digit blocks, we pair up 2 elements in the affine encrypted list. Ex: **[15,16] = 1516.** The tricky part in creating the blocks was if a number in the affine encrypted list was less than 10, there had to be a leading 0 in front of the number. So we incorporated a case where if an element < 10, a string 0 would be added in between the end of the first element (13) and beginning of second element (4). . Ex: **[13,4] = 1304**. From there we can perform the rsa encryption formula on each block. Ex: **block\*\*(e)%(p\*q).**  As a result, we have a string message that has been affine encrypted, and then rsa encrypted.

After encrypting, we now want to decrypt to give us back the original message. First we must rsa decrypt since that was the last encryption done. For rsa decryption we first ensure that an inverse of (p-1)(q-1) and e exist, which can be tested by finding that their gcf is 1. We used our inverse\_mod function from Lab 3 to calculate the inverse, d. After, we perform the rsa decryption formula on each block. Ex: **block\*\*(d)%(p\*q)**. We convert this decrypted value into a temporary string because if the decrypted value only has 2 digits, then we fill in 00 in the beginning to achieve the block of 4. Ex: **23 = 0023**. After, we split the blocks up into their previous 2 digit form through our split\_blocks function. In split\_blocks, we tested every possible block case to ensure that each number was preserved after converting them back into integers. Ex: **301 = 3, 1. 215 = 2, 15. 1506 = 15, 6.**  Now the list is prepared for affine decryption

    For affine decryption we take the inverse of **amod26** (3mod26) and perform the affine decryption formula. Ex: **inverse \*(p - b)%26.** After we can translate each affine decrypted number back to their respective letter. Our trans\_eng function takes the value(number) and retrieves the respective key (letter). After translating back, we now have our original message.

Hand Work











