Project 4 Encryption – APCSA

1. Write a function called ‘encrypt’ that takes in a String and an integer. The String is encrypted by applying different ciphers to the String. The integer is first converted to binary. Each digit of the number represents a cipher:

0: Riffle -> Find the character that appears the most in the String (not including spaces). Divide the total characters by total count of this character. Starting with the first index, move forward by this number, wrapping around from the last character to the first as is needed. This is your new index 0. The characters before this index (the substring) will now appear at the end of the substring. See example:

String example = “This is an example. The letter ‘e’ appears nine times.”;

int totalLength = example.length(); //54 characters… 54 / 9 = 6. Split String at index 6

String newString = riffle( example );

SOPln( newString ); //Printed: “s an example. The letter ‘e’ appears nine times.This i"

1: Shuffle -> Excluding spaces, add together the total numerical value of the first three characters. Any

characters between the values of [33, 126] are allowed. This range of values gives indices between [0, 94], so mod the total by 94, then add 33, and you will have a value in the range [33, 126]. Add this value to the first character, modding its value by 126 and adding 33 if the value is less than 33. Repeat this process for all characters, changing each added value for each repetition. The last characters should wrap around if needed to get the three characters after it. See example:

//These lines won’t compile. It is an example to show the math

String init = “Boo Homework”;

int valueOfFirstThree = o + o + H; //o in ascii is the value 111. H is 72. So total is 294

int nextValue = (B + 294) % 94 + 33; //B is 66 in ascii. 360 % 94 + 33 = 78 + 33 = ‘o’

String nextIteration = “ooo Homework”;

//B has been converted to ‘o’. The next little ‘o’ will be converted next.

If a String is used, such as “Test String”, and an integer number is given, such as 117, then first 117 is converted to binary, which is 64 + 32 + 16 + 4 + 1 -> 01110101. All leading zeros should be ignored, so we always begin with a shuffle. This means we would perform the following functions on the String:

String message = “Test String”;

String encodedMessage = shuffle( message ); //1

encodedMessage = shuffle( message ); //1

encodedMessage = shuffle( message ); //1

encodedMessage = riffle( message ); //0

encodedMessage = shuffle( message ); //1

encodedMessage = riffle( message ); //0

encodedMessage = shuffle( message ); //1

System.out.println(“The encoded message is: “ + encodedMessage );

This type of cipher-based system uses a private key (but this is not standard encryption) and is very math intensive for a person to do it by hand, but extremely fast for a computer to do. I highly recommend breaking the ‘encrypt’ function into the functions ‘shuffle’ and ‘riffle’, and using other functions within shuffle and riffle, such as ascii conversion.

1. Write a function called ‘decrypt’ that takes an encoded message String and the same private key, and finds the original message, by applying the algorithm in reverse. Note that this does not mean applying shuffle and riffle in the opposite order… rather, you will have to make entirely new functions to figure out how to reverse the algorithms used in the shuffle and riffle functions. Therefore, the decrypt function will be algorithmically tougher than the ‘encrypt’ function. I recommend lots of examples done on paper!

You can use Checker.java and unit test text files for these functions, but the hardest part will simply be implementing these algorithms.