```scala
object Cipher{
  /** Bit-wise exclusive-or of two characters */
  def xor(a: Char, b: Char) : Char = (a.toInt ^ b.toInt).toChar

  /** Print ciphertext in octal */
  def showCipher(cipher: Array[Char]) =
    for(c <- cipher){ print(c/64); print(c%64/8); print(c%8); print(" ") }

  /** Read file into array */
  def readFile(fname: String) : Array[Char] =
    scala.io.Source.fromFile(fname).toArray

  /** Read from stdin in a similar manner */
  def readStdin() = scala.io.Source.stdin.toArray

  /* ----- Functions below here need to be implemented ----- */

  /** Encrypt plain using key; can also be used for decryption */
  def encrypt(key: Array[Char], plain: Array[Char]) : Array[Char] = {
    var cipher = new Array[Char](plain.size); var i = plain.size
    var k = key.size
    while (i > 0) {
      i -= 1
      cipher(i) = xor(key(i%k), plain(i)) // Bit-wise exclusion of key and plaintext
    }
    cipher
  }

  /** Try to decrypt ciphertext, using crib as a crib */
  def tryCrib(crib: Array[Char], ciphertext: Array[Char]) : Unit = {
    val c = crib.size; var max = ciphertext.size - c // Max start index needed for entire crib to be found in plaintext
    var keyChars = new Array[Char](c) // Initializes keyChars to the size of the crib

    // Returns true if keyChars[0..K-j) = keyChars[j..K), for a given j
    def hasRepeat(keyChars: Array[Char], j: Int) : Boolean = {
      var yup = true; val K = keyChars.size
      var i = j
      while (yup && i < K) {
        yup = keyChars(i) == keyChars(i-j)
        i += 1
      }
      yup
    }

    var start = 0; var done = false
    var keyLength = 0
    // This loop iterates through all possible locations of the crib
    while (start <= max && !done) {
      var j = 0
      // This loop initializes keyChars to correct values
      while (j < c) {
        keyChars(j) = xor(crib(j),ciphertext(j+start))
        j += 1
      }
      // println("Start: "+start+"; KeyChars: "+keyChars.mkString(""))

      j = 0
```

```
58        while (j < c-1 && !done) { // We want at least 2 characters to be
  repeated
59          j += 1
60          done = hasRepeat(keyChars, j)
61        }
62        keyLength = j
63        start += 1
64      }
65
66      var k = 0
67      var missingFromStart = (start-1)%keyLength
68      var key = ""; var append = ""
69      while (k < keyLength - missingFromStart) { //
70        key += keyChars(k)
71        k += 1
72      }
73      while (k < keyLength) {
74        append += keyChars(k)
75        k += 1
76      }
77      key = append + key
78      println(key)
79
80      println(encrypt(key.toArray, ciphertext).mkString(""))
81    }
82
83    /** The first optional statistical test, to guess the length of the key */
84    def crackKeyLen(ciphertext: Array[Char]) : Unit = {
85      var shift = 1; val max = ciphertext.size
86      // This loop tries all possible shifts less than or equal to 30
87      while (shift <= 30) { // might consider requiring max > 30
88        var count = 0; var i = 0
89        // This loop counts the number of matches
90        while (i < max - shift) {
91          if (ciphertext(i) == ciphertext(i+shift)) count += 1
92          i += 1
93        }
94        println(shift+": "+count)
95        shift += 1
96      }
97    }
98
99    /** The second optional statistical test, to guess characters of the key.
  */
100   // Based on the idea that if there is a match, it's likely a space
101   def crackKey(klen: Int, ciphertext: Array[Char]) : Unit = {
102     var s = klen; val max = ciphertext.size; val spaceInt = ' '.toInt
103     while (s < max) {
104       var i = 0
105       while (i < max - s) { // Because we want i+s to be a valid index
106         if (ciphertext(i) == ciphertext(i+s)) {
107           val charInt = ciphertext(i).toInt ^ spaceInt // xor for the int
  versions of the relevant character and ' '
108           if (charInt >= 32 && charInt <= 127) { // Assumes plaintext only
  contains ASCII characters
109             val index = i%klen // refers to the index of the key
110             println(index.toString+" "+charInt.toChar)
111           }
112         }
113         i += 1
```

```
      }
      s += klen
    }
  }

/** The main method just selects which piece of functionality to run */
  def main(args: Array[String]) = {
    // string to print if error occurs
    val errString =
      "Usage: scala Cipher (-encrypt|-decrypt) key [file]\n"+
      "     | scala Cipher -crib crib [file]\n"+
      "     | scala Cipher -crackKeyLen [file]\n"+
      "     | scala Cipher -crackKey len [file]"

    // Get the plaintext, either from the file whose name appears in position
    // pos, or from standard input
    def getPlain(pos: Int) =
      if(args.length==pos+1) readFile(args(pos)) else readStdin

    // Check there are at least n arguments
    def checkNumArgs(n: Int) = if(args.length<n){println(errString);
sys.exit}

    // Parse the arguments, and call the appropriate function
    checkNumArgs(1)
    val command = args(0)
    if(command=="-encrypt" || command=="-decrypt"){
      checkNumArgs(2); val key = args(1).toArray; val plain = getPlain(2)
      print(new String (encrypt(key,plain)))
    }
    else if(command=="-crib"){
      checkNumArgs(2); val key = args(1).toArray; val plain = getPlain(2)
      tryCrib(key, plain)
    }
    else if(command=="-crackKeyLen"){
      checkNumArgs(1); val plain = getPlain(1)
      crackKeyLen(plain)
    }
    else if(command=="-crackKey"){
      checkNumArgs(2); val klen = args(1).toInt; val plain = getPlain(2)
      crackKey(klen, plain)
    }
    else println(errString)
  }
}


/** Test examples:

Caras-Mac:Practical1 Cara$ scala Cipher -encrypt PINGPONG santa
?' MCaras-Mac:Practical1 Cara$ ,.;*n!?;n8='4$$/4|o"(&,n
Caras-Mac:Practical1 Cara$ scala Cipher -encrypt PINGPONG santa | scala
Cipher -decrypt PINGPONG
Dear Santa, Please bring me a new bike for Christmas, love John
Caras-Mac:Practical1 Cara$ scala Cipher -encrypt SNOWMAN santa | scala Cipher
-decrypt SNOWMAN
Dear Santa, Please bring me a new bike for Christmas, love John

Caras-Mac:Practical1 Cara$ scala Cipher -crib "Dear Santa" msg
RUDOLF
```

```
Dear Santa, Please bring me a new bike for Christmas, love John

Caras-Mac:Practical1 Cara$ scala Cipher -crackKeyLen private2
1: 17
2: 12
3: 14
4: 4
5: 12
6: 8
7: 5
8: 27
9: 8
10: 6
11: 6
12: 17
13: 9
14: 6
15: 11
16: 21
17: 17
18: 14
19: 7
20: 12
21: 4
22: 9
23: 11
24: 17
25: 9
26: 3
27: 3
28: 6
29: 2
30: 5

Caras-Mac:Practical1 Cara$ scala Cipher -crackKey 8 private2 | sort -n | uniq
-c | awk '$1 > 6'
  55 0 H
  36 1 O
  28 2 G
  45 3 W
  15 4 A
  15 5 R
  21 6 T
  45 7 S

*/
```