**GALWAY-MAYO INSTITUTE OF TECHNOLOGY**

***Department of Computer Science & Applied Physics***

# Data Structures & Algorithms
# ASSIGNMENT DESCRIPTION & SCHEDULE

## *Rapid Encryption using the Four-Square Cipher*

***Note: This assignment will constitute 50% of the total marks for this module.***

## Overview

You are required to develop a Java application that is capable of encrypting and decrypting a text file or URL using a ***four-square cipher***. The four-square cipher was invented by the French cryptographer Felix Delastelle (1840-1902) and uses four 5x5 matrices arranged in a square to encrypt pairs of characters in a message. Each of the 5x5 matrices contains 25 letters (the relatively rare letter ***j*** is merged with the letter ***i***). The upper-left and lower-right quadrants (shown in green) are ***plaintext squares*** containing a standard alphabet. The upper-right and lower-left quadrants (shown in blue) are the ***ciphertext squares*** contain a mixed random alphabetic sequence. The ***four-square cipher*** is a polygraphic substitution cipher that uses the same keyword to encrypt plain-text and decrypt cipher-text, i.e. the key is symmetric.

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| A | B | C | D | E | Z | G | P | T | F |
| F | G | H | I | K | O | I | H | M | U |
| L | M | N | O | P | W | D | R | C | N |
| Q | R | S | T | U | Y | K | E | Q | A |
| V | W | X | Y | Z | X | V | S | B | L |
| M | F | N | B | D | A | B | C | D | E |
| C | R | H | S | A | F | G | H | I | K |
| X | Y | O | G | V | L | M | N | O | P |
| I | T | U | E | W | Q | R | S | T | U |
| L | Q | Z | K | P | V | W | X | Y | Z |

The encryption key is used to generate a unique sequence of letters that can be used to populate the two ***ciphertext squares***. The key can be randomly generated, specified with one keyword or specified using two different keywords (dropping duplicate letters). Any remaining spaces, i.e. a key length < 25 characters, should be filled with the remaining letters of the alphabet in order.

## Minimum Requirements

You are required to develop a Java application that can parse, encrypt and decrypt the contents of a text file (small or very large…) or URL using the four-square cipher.

- The application should contain a ***command-line menu*** that prompts the user to enter a keyword(s) and a file or URL. It should then parse the given resource line-by-line, encrypt / decrypt the text and then append to an output file.

- You must comment each method in your application stating its running time (in Big-O notation) and your rationale for its estimation.

Please note that extra marks will be given for the appropriate application of the ideas and principles we study on this course. The emphasis should be on speed (low time complexity) and a minimal amount of RAM consumption (low space complexity). There are many different ways to implement the four-square cipher.

## Deployment and Submission

- ***The project must be submitted by midnight on Sunday 8th April 2018*** as a Zip archive ***(not a rar or WinRar file)*** using the Moodle upload utility. You can find the area to upload the project under the "*Rapid Encryption using the Four-Square Cipher - (50%) Assignment Upload*" heading in the "Main Assignment" - section of Moodle.
- The name of the Zip archive should be *{id}*.zip where *{id}* is your student number.
- You must use the package name **ie.gmit.sw** for the assignment. The tableau used by the cipher is available in the class ***FourSquareCipher.java***, available on Moodle. There is also a set of small, medium and large text files, under the "*Sample Text Files for Assignment*" link that you can use to test your application.
- The Zip archive should have the following structure (do **NOT** submit the assignment as an Eclipse project):

| Name | Description |
|---|---|
| **src** | A directory that contains the packaged source code for your application. You must package your application with the namespace **ie.gmit.sw**. Your code should be well commented and explain the space and time complexity of its methods. |
| **README.txt** | A text file outlining the main features of your application. |
| **four-square.jar** | A Java Application Archive containing the classes in your application. Use the following syntax to create a JAR from a command prompt:<br>**jar –cf four-square.jar ie/gmit/sw/*.class**<br><br>Your application should be launched using the following syntax from a command prompt:<br>**java –cp ./ four-square.jar ie.gmit.sw.Runner**<br><br>**Note:** this requires that the ***main()*** method is defined inside a class called *Runner.java*. |

## Scoring Rubric

The following marking scheme will be used to score the project:

| Component | Marks (100) | Description |
|---|---|---|
| *Submission* | 5 | Zip archive correctly named and structured |
| *Compiles* | 5 | All or nothing. The Java source code compiles without errors and runs from the JAR file. |
| *README* | 5 | The README file clearly describes the application, it's features, is free from spelling and grammatical errors. |
| *Jar* | 10 | All or nothing. The JAR file is correctly named and allows the application to be launched as defined in the spec. |
| *UI* | 10 | The user interface allows files and keyword(s) to be specified and has options for output names and quit. |
| *File/URL Parser* | 10 | The parser works correctly for a file or URL and with a reasonable space and time complexity. |
| *Encrypt* | 10 | The encryption process works correctly. |
| *Decrypt* | 10 | The decryption process works correctly. |
| *Output* | 5 | The application can output the ciphertext and plaintext to file or screen. |
| *Big-O* | 10 | All methods and key steps in the application have been fully commented with Big-O running times. |

| | | |
|---|---|---|
| *Extras* | 10 | Any additional functionality that has been documented in the README. The extras must be relevant to data structures and algorithms. |

## How to Encrypt and Decrypt a Four-Square Cipher

Encryption and decryption are implemented in the following steps:

- **Step 1: Set the Keyword(s)**
  Either generate or request a keyword with a length ≥50 or two keywords with a length ≥25 containing unique characters.

  *Top Right Quadrant:* ZGPTFOIHMUWDRCNYKEQAXVSBL
  *Bottom Left Quadrant:* MFNBDCRHSAXYOGVITUEWLQZKP

  Populate the top right and bottom left quadrants with the letters from the keyword(s).

- **Step 2: Read in the Plaintext**
  Read in the plaintext from some source and convert it to either upper or lowercase to match the case of the matrix characters:
  THECURFEWTOLLSTHEKNELLOFPARTINGDAY

- **Step 3: Break up the Plaintext into Bigrams**
  *Plaintext:* THECURFEWTOLLSTHEKNELLOFPARTINGDAY
  *Bigrams:* TH EC UR FE WT OL LS TH EK NE LL OF PA RT IN GD AY
  An 'X' (or other character) can be appended to the end to ensure an even length.

- **Step 4: Encrypt Bigram**
  Locate each of the two characters in the bigram in the top left and the bottom right square respectively. Then encrypt each plain character as follows, e.g. {TH}→{ES}

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| A | B | C | D | E | Z | G | P | T | F |
| F | G | H | I | K | O | I | H | M | U |
| L | M | N | O | P | W | D | R | C | N |
| Q | R | S | **T** | U | Y | K | **E** | Q | A |
| V | W | X | Y | Z | X | V | S | B | L |
| M | F | N | B | D | A | B | C | D | E |
| C | R | H | **S** | A | F | G | **H** | I | K |
| X | Y | O | G | V | L | M | N | O | P |
| I | T | U | E | W | Q | R | S | T | U |
| L | Q | Z | K | P | V | W | X | Y | Z |

  - *Char 1:* In the top-right square, find the character at the intersection of the row of the plaintext character in the top left square with the column of the plaintext character in the bottom right square.

  - *Char 2:* In the bottom-left square, find the character at the intersection of the column of the plaintext character in the top left square with the row of the plaintext character in the bottom right square.

  Ciphertext sequence: ESPDKWUMBTWGRIESFANNWXWSWDQTHGMFTL

- **Step 5: Decrypt Bigram**
  Locate each of the two characters in the bigram in the top right and the bottom left square respectively. Then encrypt each plain character as follows, e.g. e.g. {ES}→{TH}:

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| A | B | C | D | E | Z | G | P | T | F |
| F | G | H | I | K | O | I | H | M | U |
| L | M | N | O | P | W | D | R | C | N |
| Q | R | S | **T** | U | Y | K | **E** | Q | A |
| V | W | X | Y | Z | X | V | S | B | L |
| M | F | N | B | D | A | B | C | D | E |
| C | R | H | **S** | A | F | G | **H** | I | K |
| X | Y | O | G | V | L | M | N | O | P |
| I | T | U | E | W | Q | R | S | T | U |
| L | Q | Z | K | P | V | W | X | Y | Z |

  - *Char 1:* In the top-left square, find the character at the intersection of the row of the ciphertext character in the top right square with the column of the ciphertext character in the bottom left square.

  - *Char 2:* In the bottom-right square, find the character at the intersection of the column of the ciphertext character in the top right square with the row of the plaintext ciphertext in the bottom left square.