# CS3626 Homework 03 Spring 2024

# DES

**Total Points: 25**

Be as brief as possible and use your own words when describing concepts.   SHOW ALL WORK for Questions requiring calculations and algorithms.

Q-1: Using the algorithm given below encrypt the following ASCII message using the ASCII key given below:

Message: ‘X4zr’ (You’ll need to encrypt 2 blocks to complete this message)

Algorithm: Feistel

Block Size: 16 bits

Number of Rounds: 2

Key Round 1 = ‘B’

Key Round 2 = ‘e’

Function: F = (((REi \* Key) >> 4) + REi ) mod 28

NOTE: \* is regular multiplication operator and + is LOGICAL OR Operator,

i = round number (where, when i = 0; LE0, RE0 are inputs to round 1)

>> means bit-shift right meaning binary value 01001111 >> 4 becomes 00000100: 1111 bits shifted out and zeros shifted in.

SHOW FEISTEL STRUCTURE below with the values of LEi, REi, the result of F and the result of F ⊕ LEi for each round and block of encryption.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Rounds Ri | LEi | REi | Function | F xor Lei |
| Round 1 | 01011000 00110100 | 01111010 01110010 |  |  |
| Round 2 |  |  |  |  |
| Output |  |  |  |  |

What is the transmitted ciphertext (provide in hexadecimal) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**6 points**

**FYI, YOUR PROJECT WILL IMPLEMENT FEISTEL using Galois Field Math in Function F, should you wish to start Feistel Implementation now with a regular function and later replacing F with Galois F.**

Q-2: What is the result of the following inputs to these DES (sort-of) functions:

Input[16-bits]: 0x3C1D [MSB = bit position 1 left to right]

Expansion Box: First input bit position [14] represents output position [1], that is counting from 1 not 0.

14 16 3 6 8 13

5 15 13 9 14 7

8 11 4 12 10 2

10 2 1 7 5 12

What is output (in hexadecimal) of the expansion box function: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**2 points**

Q-3: The output above is broken into 6-bit segments and sent from left to right to the following four DES S-Boxes in order S1, S2, S3 and S4.

Text

Description automatically generatedText

Description automatically generated

Text

Description automatically generatedText

Description automatically generated

**What is the resulting Output (in hexadecimal):\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**2 points**

**Q-4: The output above is then circular bit shifted left by 5 bits.**

**What is the resulting shifted Output (in hexadecimal):\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**2 points**

**Q-5: Upgrade your Caesar algorithm to Vigenere to encrypt and decrypt. Also upgrade it to allow changing the hard coding of the ASCII alphabet (if not already done). E.g. set a variable (however your language does this) to define an alphabet. As an example, for C++ std::vector or std::string is recommended, such as:**

std::string alphabet = “abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789”;

where a = 0, b = 1, c = 2 etc…

--- being changed and rebuilt to use this alphabet ---

std::string alphabet = “0123456789abcdefghijklmnopqrstuvwxyz”;

where ‘0’ = 0, ‘1’ = 1, ‘a’ = 10 etc…

**Upgrade your vigenere code to read from an input file and an output file. These can be supplied as arguments or hard-coded (but arguments will work better for future assignments), e.g.**

vigenere <input\_file> <key> <output\_filename> <e = encrypt | d = decrypt>

Example execution on command line: ./vigenere plaintext.txt mykeyisthis cyphertext.txt e

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Example BEGIN Example \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Using the alphabet: std::string alphabet = “abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789”

**NOTE: PASS through any values not in your alphabet. For example: if KEY = “box”**

Message: I have a dot.number(1) Comma, Colon: and parenthesis

Cyphertext result: J vxws x eCQ.oIJcsO(2) QLnAx, DCIpB: xor MbFBoHEfGFt

Spaces are not in alphabet and pass through, punctuation and spaces do as well in this example.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Example END Example \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

------------------ BEGIN ASSIGNMENT REQUIRED RESPONSE INSTRUCTIONS --------------------

**Response 1: Using plain1.txt as input file-------------------------------------------------------------------------------------**

Alphabet “abcdefghijklmnopqrstuvwxyz\_ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789”

The underscore above ( \_ ) between z and A is meant to represent the SPACE character (0x20).

Use Key: Key

**Any plaintext value not in alphabet is just copied through untouched.**

**Any key value not in alphabet should generate an error and exit.**

**Key is indexed only on valid alphabet characters, a character outside the alphabet does not cause key to move to next character in key.**

Using this key = “Key” and alphabet above **encrypt** plain1.txt and provide the cypher output here:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Hint: do you see the text?: ,D1nEXB4

**Response 2: Using cypher2.txt as input file.----------------------------------------------------------------------------------**

Next, change the alphabet to: std::string alphabet = “aeiouycdx\_IVXLCDMK012”; // the \_ here is a SPACE (0x20)

Using this alphabet and key = “Key” **decrypt** cypher2.txt and provide the plaintext output here:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Hint: CHROMA V2

Provide source-code per instructions below. Keep this code it will be revisited.

**8 points**

**More Examples: here is another short example, see if your algorithm matches,**

**For vigenere, when the letter is not in the alphabet for the key, exit the program and error out. When a letter in the plaintext is not in the alphabet, pass it through but do not count it as a character against the key (that is keep the current key index, skipping over non alphabet characters in plaintext, and use it for the next valid alphabet character). Do this both for encryption and decryption [since passthrough can put non-alphabet characters in cyphertext].**

**key = box alphabet = [a-z] { lowercase only in other words}**

**message = "GrancisBacons"**

**Encryption is then:**

**\*boxbo\*xboxbo K // where \* means skip over plaintext and don't use up that key element either.**

**GranciSBacons +P // Input has capital letters that are just passed through.**

**\_\_\_\_\_\_\_\_\_\_\_\_\_**

**GsokdwSBxdckt =C // The G S B are passed through and the key does not index up on those chars.**

**that is to say if only lower case are counted:**

**boxboxboxb K**

**ranciacons +P**

**\_\_\_\_\_\_\_\_\_\_\_**

**sokdwxdckt =C**

**Q-6: Upgrade your frequency analyzer to provide the a sorted from most frequent to least frequent output characters. Allow all possible extended-ASCII character values from 0 – 255. Note: 0 = NULL.**

**Plain Text File = SherlockHolmes.txt [see D2L files]**

**Provide the frequency spectral output of SherlockHolmes.txt here with counts sorted (either descending or ascending).**

**As an example phrases.txt produces the sorted spectrum:**

**(Note: 0 results were suppressed and in this example values were left out to demonstrate here. If character was unprintable, it was replaced with hexadecimal value e.g. 0x00a, 0x00d - you are not required to do this but it helps make the output less messy)**

**Sample Output of Frequency:**

**- 3981 of 21964**

**e - 2297 of 21964**

**a - 2113 of 21964**

**i - 1207 of 21964**

**t - 1070 of 21964**

**n - 1031 of 21964**

**0x00a - 990 of 21964**

**0x00d - 990 of 21964**

**.**

**. (Median results omitted for clarity)**

**.**

**. - 2 of 21964**

**B - 1 of 21964**

**C - 1 of 21964**

**M - 1 of 21964**

**Most common: 3981(' ' = 20) - 21964 of 21964**

Provide source-code per instructions below. Keep this code it will be revisited.

**5 points**

**Submission Guidelines:**

* No handwritten submission is accepted, always submit answers as text within this or similar document file with any support images embedded in the file.
* **EXCEPTION**: If asked for source code implementation you can submit those individually and as separate files in ASCII format in their original file format .cpp, .java, .py, .cs etc. or even as a .txt file will be acceptable. Do not insert code into the submission document file. It ruins spacing which makes .python and some languages (perl, awk etc.) difficult to test build.
* Do not submit ZIP files… ever… for anything in D2L. The system is extremely unhelpful with regards to those filetypes and grading.
* You may include your freehand drawing/image and handwritten scans in the submission. However, the writing and images must be clearly legible. Though, it is best to present non-handwritten submissions, generally, as is done in the professional setting.
* If asked, show all work/calculations/graphs etc. in the determination of the problem.
* **Please complete your entire work in a single Word Document and Save the file as: yournetid\_CS3502\_Assignment01.docx (e.g. ogarcia5\_CS3502\_Assignment01.docx.) and upload your file in D2L.**
* Please observe the submission due date and time. After the due date there is a 50% penalty for the next 24 hours. Any submission after 24 hours of the due date will be graded at 0%.
* If you include a reference or an image taken from other sources, please cite them appropriately. APA is preferred but cite them so they can be found. **NOTE: verbatim copying or even paraphrasing is plagiarism so if the source used constitutes your answer rather than simply *supporting* the answer, it will be considered invalid. This is especially true of source code implementation answers.**
* If you resubmit, please make sure to attach the file again. Your latest submission before the due date will be the one graded.