INF243-Mandatory Assignment 4

Submission Deadline: April 30th, 2023

Instructions for the assignment:

- This assignment has 5 pages and accounts for 10 points for your final grade
- Prepare a PDF file for your answers
  - you can use Latex (see manual at this link) as the text editor which compiles to a nice PDF file
  - this assignment is more implementation-oriented
  - You are suggested to use SageMath to work on the questions
  - make sure to properly comment your source code.
  - If you answer questions with SageMath, you can provide necessary text for each question and export it directly as a PDF file via Latex

Q1. Basic Reed-Solomon (RS) Codes [1 pts]

Let α be a primitive element of $GF(2^4)$ satisfying $p(\alpha) = \alpha^4 + \alpha + 1 = 0$ and C be a triple-error-correcting, narrow-sense, RS code of length 15 over $GF(2^4)$.

(i) Suppose the RS code is given in BCH representation and the received word is

$$r(x) = \alpha^4 x^{12} + \alpha^7 x^9 + \alpha^2 x^7 + x^6 + \alpha^{10} x^5 + \alpha x^4 + \alpha^{12} x^3 + \alpha^4 x^2 + \alpha^{13}.$$

Provide the generator polynomial of C and calculate the syndrome S_i for $i = 1, \ldots, 6$.

(ii) Assume the RS code is given in polynomial representation with its codeword given as

$$\mathbf{c} = (c_0, c_1, \ldots, c_{14}) = (f(\beta_0), f(\beta_1), \ldots, f(\beta_{14})),$$

where $\beta_i = \alpha^i$ for $i = 0, 1, \ldots, 14$. Suppose a receive word is given by

$$\mathbf{r} = (r_0, r_1, \ldots, r_{14}) = (0, 0, 0, \alpha^4, 0, 0, \alpha^7, 0, \alpha^2, 1, \alpha^{10}, \alpha, \alpha^{12}, \alpha^4, \alpha^{13}).$$

Use the Berlekamp-Welch algorithm (notes: 1, 2) to find the sent codeword $\mathbf{c}$.

Q2. RS Codes [2 pts]

Let α be a primitive element of $GF(2^8)$ satisfying $p(\alpha) = \alpha^8 + \alpha^4 + \alpha^3 + \alpha^2 + 1 = 0$. Implement the following functions for a $[255, 128]$ RS code C over $GF(2^8)$ with generator polynomial

$$g(x) = (x - 1)(x - \alpha)(x - \alpha^2) \ldots (x - \alpha^{127})$$

- `encodeRS`($\mathbf{m}$): this functions takes the 128-byte message and encode it to a codeword $\mathbf{c}$ with a systematic encoding as

$$\mathbf{m} = (m_0, m_1, \ldots, m_{k-1}) \Longrightarrow \mathbf{c} = (m_0, \ldots, m_{k-1}, c_k, \ldots, c_{n-1})$$

- `decodeRS`($\mathbf{r}$): this function uses the Sugiyama decoding method to output a codeword $\mathbf{c} \in C$ or outputs a message "Failure of unique decoding" when $d(\mathbf{c}, \mathbf{r}) > 63$.

Q3. Goppa codes were used to build the McEliece cryptosystem (whose optimized version `Classic McEliece` entered in the Round 4 Finalist in the NIST's standardization on post-quantum cryptography). The follow questions aim to familiarize you with the Goppa code and the original McEliece cryptosystem. [4 pts]

Let m be a positive integer, $n = 2^m$ and $g(x)$ be a binary irreducible polynomial of degree t satisfying $g(a) \neq 0$ for any $a \in GF(2^m)$. Define the binary Goppa code as follows

$$\Gamma = \left\{ \mathbf{c} = (c_0, c_1, \ldots, c_{n-1}) \in GF(2)^n \mid \sum_i \frac{c_i}{x - \beta_i} \bmod g(x) \equiv 0 \right\}$$

where $\beta_0, \ldots, \beta_{n-1}$ are elements in $GF(2^m)$.

The original McEliece crytosystem basically proceeds with the following steps:

> **Key Generation:** private key (S, G, P) and public key $G' = SGP$
>
> **Encryption:** for a message $\mathbf{m}$, its ciphertext $\mathbf{c} = \mathbf{m}G' + \mathbf{e}$, where $\mathbf{e}$ has Hamming weight t
>
> **Decryption:** decode $\mathbf{m}S$ from $\mathbf{c}$ with the decoding of Goppa code and then recover the message $\mathbf{m}$

Suppose $m = 6$ and α is a primitive element of $GF(2^6)$ satisfying $p(\alpha) = \alpha^6 + \alpha^4 + \alpha^3 + \alpha + 1 = 0$ and $g(x) = x^7 + x + 1$.

- Build an encoder and decoder with Patterson's algorithm in SageMath for a binary Goppa code with the above parameters.
- Implement the **Key Generation**, **Encryption**, **Decryption** steps of McEliece cryptosystem based on your implemented Goppa encoder and decoder.

  Use your implementation to encrypt the message:

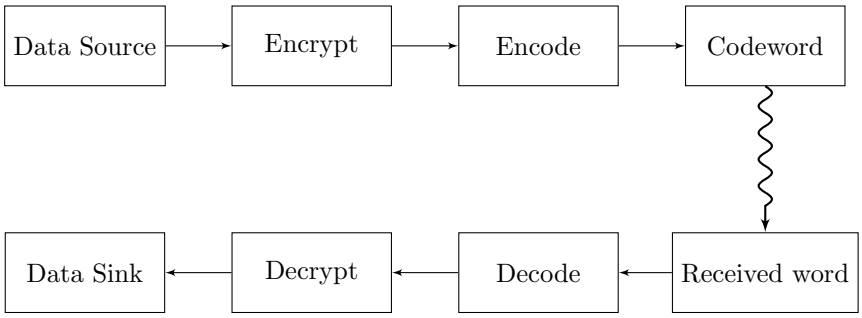
  `TIME FLIES QUICKLY`

  and decrypt the ciphertext based on the Goppa decoder

  In your encryption, you skip the space and convert each uppercase letter to a 8-bit binary string with ASCII encoding. In this way the message is converted to 128 bits. Divide the 128-bit binary string into two 64-bit blocks for Goppa encoder and decoder.

Note: SageMath implemented Goppa encoder, but it uses the syndrome decoder which is a lot slower than Patterson's algorithm. You are suggested to learn the implementation in the master thesis by Roering Christopher. However, you cannot directly use his implementation.

Q4. Simulate a simple communication model with McEliece cryptosystem and RS code. [3 pts]



Suppose Alice wants to send Bob some secret message in an insecure and noisy environment, they can use the above communication model to achieve secure and reliable communication by following the steps below:

1. Bob chooses the McEliece cryptosystem in Q4 and makes his public key G' available to Alice
2. Alice wants to send the following message

   `WHENEVER WE TRANSMIT SENSITIVE INFORMATION OVER THIS CHANNEL, THERE IS A POTENTIAL THAT AN EAVESDROPPER COULD ACCESS THE CONTENT`

   She converts the message into 1024 bits with python/Sagemath commands and divides the data into 16 blocks of 64 bits. She encrypts each block with McEliece cryptosystem using Bob's public key G' (in the same way as in Q4)
3. Alice encodes the 128-byte ciphertext with the RS code in Q3, and sends Bob a 255-byte codeword

   Suppose in the transmission, the codeword is distorted and 63 positions in the received word are erroneous, where the positions and erroneous values are randomly chosen. When Bob receives the data, he proceeds the following steps

4. Bob decodes the received word (containing 63 erroneous positions)
5. Bob divides the codeword into 16 blocks, each block has 64 bits
6. Bob decrypts each 64-bit block, converts them to letters and join them together

NB: In Q5, it integrates your implementations of RS codes and McEliece cryptosystem. In the task, the conversion between binary string and text

is not an important part. You can use the following conversion in your implementation

```python
def text2bin(msg):
    return "".join(f"{ord(i):08b}" for i in msg)

def bin2text(bin_s):
    return "".join([chr(int(bin_s[i:i+8],2)) for i in range(0,len(bin_s),8)])

msg = "Error-correcting codes are crucial for wireless communications!"

data = text2bin(msg)
print(data)

msg1 = bin2text(data)
print(msg1)
```

```
01000101011100100111001001101111011100100010110101100011011011110111001001110010
01100101011000110111010001101001011011100110011100100000011000110111011101100100
01100101011100110010000001100001011100100110010100100000011000110111001001110101
01100011011010010110000101101100001000000110011001101111011100100010000001110111
01101001011100100110010101101100011001010111001101110011001000000110001101101111
01101101011011010111010101101110011010010110001101100001011101000110100101101111
01101110011100110010000 1
Error-correcting codes are crucial for wireless communications!
```