UNIVERSIDAD
DE GRANADA

Escuela Técnica Superior de Ingeniería Informática y de
Telecomunicaciones y Facultad de Ciencias

Doble Grado en Ingeniería Informática y Matemáticas

TRABAJO DE FIN DE GRADO

El criptosistema de McEliece usando códigos Reed-Solomon generalizados

Presentado por:
Higinio Paterna Ortiz

Curso académico 2024-2025

El criptosistema de McEliece usando códigos Reed-Solomon generalizados

Higinio Paterna Ortiz

Higinio Paterna Ortiz *El criptosistema de McEliece usando códigos Reed-Solomon generalizados*. Trabajo de fin de Grado. Curso académico 2024-2025.

Responsable de tutorización

Gabriel Navarro Garulo
Departamento de Álgebra

Doble Grado en Ingeniería Informática y Matemáticas

Escuela Técnica Superior de Ingeniería Informática y de Telecomunicaciones y Facultad de Ciencias

Universidad de Granada

Declaración de originalidad

D./Dña. Higinio Paterna Ortiz

Declaro explícitamente que el trabajo presentado como Trabajo de Fin de Grado (TFG), correspondiente al curso académico 2024-2025, es original, entendido esto en el sentido de que no he utilizado para la elaboración del trabajo fuentes sin citarlas debidamente.

En Granada a 15 de junio de 2025

Fdo: Higinio Paterna Ortiz

Summary

Keywords: Cryptography, error-correcting codes, computational complexity, algebra, attack.

Every week we happen to hear about a new advance in quantum computing, with computers which have a qubits capacity that does not stop growing. The theoretical algorithms thought to be implemented when this technology would be feasible, such as Shor's algorithm, appear to be closer than ever to breaking the mathematical problems on which the vast majority of cryptosystems running are based. Because of that, there comes the need for other kinds of problems, harder to solve by a quantum machine, on which cryptography can rely. The discipline in charge of that search is called post-quantum cryptography.

Between its different lines of research, in this Bachelor's Thesis we have chosen the code-based one. This approach relies on error-correcting codes, a field that tries to solve the issue of errors added to the data while transmitted over a noisy channel. Within these codes, we will focus on the linear ones, as their algebraical structure allows us to operate with them in terms of linear algebra, that is, with matrices, vectors and their operations. Their crucial property for the security of the cryptosystems will be the NP-Completeness of the binary linear-code decoding's problem, which is precisely the problem which we were looking for. In order to introduce all these notions, we will have to do some preliminary work about algebra and computational complexity.

One of the advantages that we have mentioned about linear codes is the possibility of working with them in terms of matrices and vectors. However, an even easier way to manipulate codes would be through polynomials, with their well-known sum and product operations. For this purpose, we will introduce the cyclic codes. Among the subfamilies of these codes, the BCH codes are particularly interesting, because we can define the lower bound that we want on the minimum distance. Furthermore, if we impose a BCH code to have a length equal to the size of the field minus 1, we obtain a Reed-Solomon code. Should we add a vector of weights to the previous code, then we obtain a new family, the generalized Reed-Solomon codes, which are no longer cyclic, but they still preserve the minimun distance properties of the Reed-Solomon codes and allow for more flexibility.

Once we will have studied the linear codes and some of their subfamilies, we will review briefly the basis of cryptography, introducing its common classification between symmetric and asymmetric. The one most relevant for our cause is the asymmetric one, where we can place the McEliece cryptosystem.

The main work of the project lies on the last two chapters. At first, we will introduce the McEliece cryptosystem based on Goppa codes, which is one of the cryptosystems presented to the contest organized by NIST (National Institute of Standards and Technology) in order to standardize robust post-quantum cryptosystems. Nevertheless, we will expose some disadvantages of this scheme that will make us to contemplate other alternatives. There are several proposals in the literature, but the one that this project focuses on is the use of

generalized Reed-Solomon codes instead of Goppa ones. The three main algorithms that build a cryptosystem will be implemented: generation of keys, encryption and decryption. Although apparently the main pitfalls of the original proposal will seem to be solved, the structure and some properties of the generalized Reed-Solomon codes will put in trouble the security of the cryptosystem. Some of the vulnerabilities are exploded by attacks like the Sidelnikov-Shestakov one. Under certain conditions, we will prove both theoretically and practically that this attack can break the cryptosystem, deciphering any ciphertext being sent without possessing the private key, in reasonable time.

Resumen

Palabras clave: Criptografía, códigos correctores de errores, complejidad computacional, álgebra, ataque.

Cada semana escuchamos acerca de un nuevo avance en computación cuántica, con ordenadores que tienen una capacidad de cúbits que no para de crecer. Los algoritmos teóricos pensados para implementarse cuando esta tecnología fuera viable, como el algoritmo de Shor, parecen estar más cerca que nunca de romper los problemas matemáticos en los que la mayoría de criptosistemas en funcionamiento están basados. Debido a ello, surge la necesidad de otros tipos de problemas más difíciles de resolver por ordenadores cuánticos, en los cuales la criptografía pueda confiar. La disciplina encargada de la búsqueda de dichos problemas es conocida como criptografía post-cuántica.

Entre sus principales líneas de investigación, en este Trabajo de Fin de Grado hemos escogido la basada en teoría de códigos. Este enfoque se basa en los códigos correctores de errores, una disciplina que trata de resolver la problemática de los errores añadidos a la información que viaja a través de un canal con ruido. Dentro de estos códigos, nos centraremos en los lineales, ya que su estructura algebraica nos permite trabajar con ellos en términos del álgebra lineal, es decir, con matrices, vectores y las operaciones entre ellos. Su propiedad más relevante de cara a la seguridad de los criptosistemas será la NP-Completitud del problema de decodificación de un código lineal binario, que es precisamente el problema que estábamos buscando. Para introducir todos estos conceptos, tendremos que hacer algo de trabajo preliminar sobre álgebra y complejidad computacional.

Una de las ventajas que hemos mencionado sobre los códigos lineales es la posibilidad de trabajar con ellos en términos de matrices y vectores. Sin embargo, una manera incluso más sencilla de manejar los códigos sería a través del uso de polinomios, con sus operaciones bien conocidas de suma y multiplicación. Con este fin, introduciremos los códigos cíclicos. Entre las subfamilias de estos códigos, los códigos BCH son particularmente interesantes, ya que podemos definir la cota inferior que queramos sobre la distancia mínima. Si además imponemos a un código BCH que tenga una longitud igual al tamaño del cuerpo menos 1, obtenemos un código Reed-Solomon. Añadiendo un vector de pesos al código anterior obtendremos una nueva familia, los códigos Reed-Solomon generalizados, los cuales dejan de ser cíclicos, pero conservan las propiedades de la distancia mínima de los códigos Reed-Solomon y permiten mayor flexibilidad.

Una vez que hayamos estudiado los códigos lineales y algunas de sus subfamilias, repasaremos brevemente las bases de la criptografía, introduciendo su clasificación común en simétrica y asimétrica. La más pertinente para nuestro estudio es la asimétrica, donde podemos ubicar al criptosistema de McEliece.

El trabajo principal del proyecto se concentra en los últimos dos capítulos. En primer lugar,

introduciremos el criptosistema de McEliece basado en códigos Goppa, que es uno de los criptosistemas presentados al concurso organizado por el NIST (Instituto Nacional de Estándares y Tecnología, son sus siglas en inglés) para estandarizar criptosistemas post-cuánticos robustos. Sin embargo, expondremos algunos inconvenientes de este esquema que nos harán contemplar otras alternativas. Hay muchas propuestas en la literatura, pero en la que se centra el trabajo es en el uso de códigos Reed-Solomon generalizados en vez de Goppa. Los tres principales algoritmos que conforman un criptosistema serán implementados: generación de claves, cifrado y descifrado. Aunque aparentemente las principales desventajas de la propuesta original parecerán estar resueltas, la estructura y algunas propiedades de los códigos Reed-Solomon generalizados pondrán en entredicho la seguridad del criptosistema. Algunas de las vulnerabilidades son explotadas por ataques como el de Sidelnikov-Shestakov. Bajo ciertas condiciones, probaremos tanto teórica como prácticamente que este ataque puede romper el criptosistema, descifrando cualquier texto cifrado que sea enviado sin disponer de la clave privada, en un tiempo razonable.

Índice general

Introducción

El fin principal de este trabajo ha sido desde el primer momento la presentación e implementación del criptosistema de McEliece basado en códigos Reed-Solomon generalizados, así como del principal ataque en la literatura que prueba su inseguridad, el ataque de Sidelnikov-Shestakov. Ambos objetivos han sido alcanzados, pues se ha probado la corrección desde el punto de vista tanto teórico como práctico (vía las validaciones cuyos resultados se reflejan al final de los capítulos 5 y 6). Como objetivos secundarios, aunque imprescindibles para la comprensión de los principales, se propusieron y se han llevado a cabo los siguientes:

- Introducción del concepto de código lineal y de sus propiedades más importantes, haciendo especial énfasis sobre aquellas que justifican el uso de los códigos en criptografía.
- Presentación de las subfamilias de códigos relacionadas con los Reed-Solomon generalizados, yendo de la más genérica a la más específica.
- Implementación de un codificador-decodificador usando códigos Reed-Solomon generalizados.
- Repaso de los fundamentos de la criptografía, sus tipos elementales y de su línea de investigación conocida como criptografía post-cuántica.

Las bases teóricas sobre códigos desarrolladas en los capítulos 2 y 3 usan como fuente principal [yVP03]. Como referencia para el diseño del criptosistema de McEliece hemos utilizado el trabajo recogido en [D.E07], mientras que para el ataque, aunque el trabajo original de Sidelnikov y Shestakov es [ySS92], debido a la falta de ciertos detalles de cara a la implementación hemos recurrido a [Wie10].

1 Preliminares

1.1. Complejidad Computacional

La siguiente sección se basa en la referencia [Mor20].

Una de las métricas fundamentales a la hora de analizar un criptosistema es la "dificultad" a la hora de realizar sus funciones básicas (encriptación y desencriptación) y a la hora de que un posible atacante pueda romper el sistema. Para ello, es necesario primero formalizar lo que entendemos por complejidad de un problema computacional. Comencemos por la definición de la unidad básica de trabajo en computabilidad y complejidad computacional.

Definición 1.1.1 (Máquina de Turing)**.** Una máquina de Turing (MT) determinista es una séptupla (Q, A, B, δ, q_0, #, F) en la que

- Q es un conjunto finito, al que denotamos como estados.
- A es un conjunto finito, al que denotamos como alfabeto de entrada.
- B es un conjunto finito, al que denotamos como alfabeto de símbolos de la cinta, e incluye a A.
- δ es una función $\delta : Q \times B \to Q \times B \times \{I, D\}$, de manera que o bien $\delta(q, b) = \emptyset$, o bien $\delta(q, b) = (p, c, M)$, $\forall q \in Q, \forall b \in B, p \in Q, c \in B, M \in \{I, D\}$. A esta función la denotaremos como función de transición.
- q_0 es un elemento especial de Q al que denotaremos como estado inicial.
- # es un símbolo especial de $B \setminus A$ al que llamaremos símbolo blanco.
- F es un conjunto finito, al que llamaremos estados finales.

Para ilustrar esta definición, veamos un ejemplo sencillo de MT.

Ejemplo 1.1.1. Consideremos la MT $M = (\{q_0, q_1, q_2, q_3, q_4\}, \{0, 1\}, \{0, 1, X, Y, \#\}, \delta, q_0, \#, \{q_4\})$ donde las transiciones no nulas son las siguientes:

$$\begin{array}{ll} \delta(q_0, 0) = (q_1, X, D) & \delta(q_0, Y) = (q_3, Y, D) \\ \delta(q_1, 0) = (q_1, 0, D) & \delta(q_1, 1) = (q_2, Y, I) \\ \delta(q_1, Y) = (q_1, Y, D) & \delta(q_2, 0) = (q_2, 0, I) \\ \delta(q_2, X) = (q_0, X, D) & \delta(q_2, Y) = (q_2, Y, I) \\ \delta(q_3, Y) = (q_3, Y, D) & \delta(q_3, \#) = (q_4, \#, D) \end{array}$$

Veamos la ejecución de un par de transiciones de esta MT usando la figura 1.1. El concepto de cinta de una MT será redundante en el resto del apartado, ya que nos ayuda considerablemente a comprender el funcionamiento de estas máquinas. Comenzaríamos en el estado inicial q_0. Leemos un 0 en la cinta, de manera que pasaríamos al estado q_1, escribiríamos X

en la posición donde nos encontramos y nos moveríamos una casilla a la derecha. Leemos otro 0, permaneciendo ahora en el mismo estado, no sobreescribiendo la posición actual y avanzando otra casilla a la derecha.

Presentemos ahora un caso más general de MT que vamos a necesitar en posteriores definiciones.

Definición 1.1.2 (Máquina de Turing No Determinista)**.** Una máquina de Turing no determinista (MTND) es una MT en la que ampliamos la definición de δ, cuyo codominio pasa a ser $\mathcal{P}(Q \times B \times \{I, D\})$ (donde $\mathcal{P}(\cdot)$ denota partes de un conjunto), de manera que ahora será válido $\delta(q, b) = \{(q_1, b_1, M_1), ..., (q_k, b_k, M_k)\}$.

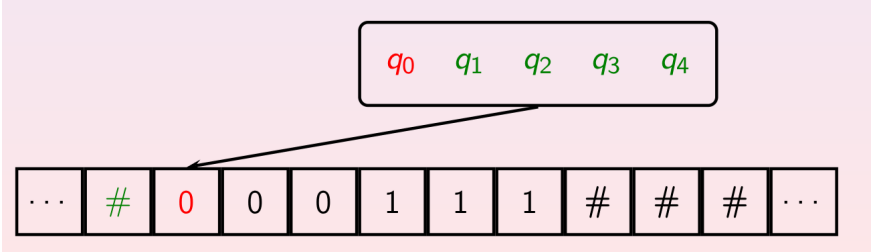

Figura 1.1: Cinta de una MT [Mor20]

El concepto de función calculada por una MT también nos servirá más adelante.

Definición 1.1.3. Dada una MT $M = (Q, A, B, \delta, q_0, \#, F)$, llamaremos función calculada por M a una función $f : D \to B^*$, con $D \subseteq A^*$ el conjunto de entradas para los que M pasa por un estado de parada, tal que $f(u)$ representa el contenido de la cinta cuando M se encuentra en uno de estos estados, excluyendo los símbolos en blanco, $\forall u \in D$. A^* denota el conjunto de palabras que podemos formar usando el alfabeto A, incluida la palabra vacía.

Pasemos a definir una serie de conceptos clave en el contexto de las MTs.

Definición 1.1.4 (Configuración)**.** Una configuración de una MT es una tripleta (q, w_1, w_2) donde

- q es el estado actual de la máquina.
- w_1 es la representación de la parte de la palabra que hay a la izquierda de la posición del cabezal de lectura (puede ser vacío). Esta representación se obtiene eliminando la sucesión infinita de blancos a la izquierda de la última casilla distinta de blanco.
- w_2 es la representación de la parte de la palabra que se obtiene empezando en el cabezal de lectura y luego continuando hacia la derecha. No puede ser vacío. Esta representación se obtiene eliminado la sucesión infinita de blancos a la derecha de la última casilla distinta de blanco.

Definición 1.1.5 (Configuración inicial)**.** Si $u \in A^*$, la configuración inicial de la MT $(Q, A, B, \delta, q_0, \#, F)$ asociada a esta palabra es (q_0, ϵ, u), siendo $(q_0, \epsilon, \#)$ si $u = \epsilon$, con ϵ la palabra vacía.

Definición 1.1.6 (Paso de cálculo a la izquierda)**.** Si $\delta(q,a) = (p,b,I)$ entonces decimos que de la configuración $(q, c_1...c_n, ad_2...d_m)$ llegamos en un paso de cálculo a la configuración $(p, c_1..c_{n-1}, c_nbd_2...d_m)$, lo que se denota como $(q, c_1...c_n, ad_2...d_m) \vdash (p, c_1...c_{n-1}, c_nbd_2...d_m)$, suponiéndose:

- Si $c_1...c_n = \epsilon$, entonces $c_1...c_{n-1} = \epsilon$ y $c_n = \#$.
- Se eliminan los blancos a la derecha de la palabra $c_nbd_2...d_m$ excepto el primero, si toda la palabra está formada por blancos.

Definición 1.1.7 (Paso de cálculo a la derecha)**.** Si $\delta(q,a) = (p,b,D)$ entonces decimos que de la configuración $(q, c_1...c_n, ad_2...d_m)$ llegamos en un paso de cálculo a la configuración $(p, c_1..c_nb, d_2d_3...d_m)$ lo que se denota como $(q, c_1...c_n, ad_2...d_m) \vdash (p, c_1..c_nb, d_2d_3...d_m)$, considerándose:

- Si $m = 1$ entonces $d_2d_3...d_m = \#$.
- Se eliminan todos los blancos a la izquierda en $c_1..c_nb$.

Definición 1.1.8 (Relación de pasos de cálculo)**.** Si R y R' son configuraciones de una MT $(Q, A, B, \delta, q_0, \#, F)$ se dice que desde R se llega en una sucesión de pasos de cálculo a R', lo que se denota como $R \overset{*}{\vdash} R'$, si y solo si existe una sucesión finita de configuraciones $R_1, ..., R_n$ tal que $R = R_1, R' = R_n$ y $R_i \vdash R_{i+1}, \forall i < n$.

Definición 1.1.9 (Lenguaje aceptado)**.** Si M es una MT, entonces el lenguaje aceptado es el conjunto de palabras $L(M)$ tales que $u \in L(M)$ si y solo si existen $w_1, w_2 \in A^*$ y $q \in F$ tales que $(q_0, \epsilon, u) \overset{*}{\vdash} (q, w_1, w_2)$.

Estrechamente ligado con el concepto de lenguaje aceptado está el de problema.

Definición 1.1.10 (Problema)**.** Un problema $PROBLEMA(x)$ o $PROBLEMA$ consta de:

- Un conjunto X al que llamaremos entradas.
- Un conjunto Y al que llamaremos soluciones.
- Una aplicación $R : X \times Y \to \{0,1\}$ que representa la relación que debe de existir entre las entradas y las soluciones, de manera que $R(x,y) = 1$ significará que la solución $y \in Y$ es una solución para la entrada $x \in X$, y $R(x,y) = 0$ en caso contrario.

Veamos un primer ejemplo para comprender la definición de problema.

Ejemplo 1.1.2. Búsqueda de ciclo hamiltoniano en un grafo:

- Entradas X: conjunto formado por grafos arbitrarios.
- Conjunto Y: sucesiones de aristas adyacentes $(a_1, ..., a_k)$, es decir, dadas $a_i = (v_l, v_m)$ y $a_{i+1} = (v_q, v_s)$, $v_m = v_q$, $\forall i \in \{1, ...k-1\}$, donde v_l, v_m, v_q, v_s son vértices arbitrarios.
- La relación que debe de haber entre las entradas y las soluciones es que dado un grafo G, $(a_1, ..., a_k)$ debe ser un subconjunto de sus aristas, el conjunto de todos los vértices por los que pasan dichas aristas no debe contener repeticiones y debe ser igual al conjunto de vértices de G, y dadas $a_1 = (v_0, v_1), a_k = (v_{k-1}, v_k)$, $v_0 = v_k$.

Existen distintos tipos de problemas, aunque en nuestro estudio los que más nos van a interesar son los de decisión.

Definición 1.1.11 (Problema de Decisión)**.** Un problema de decisión es aquel en el que $Y = \{SI, NO\}$ y $\forall x \in X, !\exists y \in Y : R(x, y) = 1$.

El problema del ejemplo 1.1.2 puede ser reformulado como un problema de decisión.

Ejemplo 1.1.3. Dado un grafo arbitrario, determinar si contiene un ciclo hamiltoniano o no.

Habitualmente solemos codificar al conjunto X en un lenguaje. Por tanto, podemos establecer un nexo entre lenguajes y problemas de decisión.

Definición 1.1.12 (Lenguaje asociado a un problema de decisión)**.** Dado un problema de decisión PROBLEMA, el lenguaje asociado a dicho problema se define como el conjunto $L(PROBLEMA(x)) = \{x \in A^* : PROBLEMA(x) = 'SI'\}$. Un lenguaje L sobre un alfabeto A siempre define también un problema de decisión: dada $x \in A^*$ determinar si $x \in L$.

Observación 1.1.1. Esta definición nos va a permitir hablar de problemas de decisión y de lenguajes de forma intercambiable.

Pasemos a hablar ya de complejidad en tiempo, yendo directamente a la definición de clase de complejidad no determinista.

Definición 1.1.13 (Clases no deterministas)**.** Una MTND tiene complejidad $f(n)$ en tiempo si y solo si para una entrada x de longitud n todos los posibles pasos de cálculo de la máquina terminan en $f(n)$ pasos. Denotaremos por $NTIEMPO(f)$ a la clase de todos los lenguajes aceptados por una MTND en tiempo $O(f(n))$.

De entre todas las clases no deterministas, nos va a interesar especialmente la polinómica, que se puede definir como $NP = \cup_{j>0} NTIEMPO(n^j)$. Antes de presentar una familia muy importante dentro de los lenguajes de NP, debemos ocuparnos del concepto de reducción.

Definición 1.1.14 (Reducción)**.** L_1 es reducible a L_2 si y solo si existe una MT determinista que, de manera eficiente, calcula una función $R : A^* \to B^*$ que verifica $x \in L_1 \leftrightarrow R(x) \in L_2$.

Observación 1.1.2. El concepto de eficiente usado en la definición previa es lo que en computabilidad se denota como reducción en espacio logarítmico. La idea subyacente es que la MT debe consumir en la reducción una cantidad de espacio pequeña.

Finalmente, llegamos a la definición clave de esta sección.

Definición 1.1.15 (Lenguaje NP-completo)**.** Un lenguaje es NP-completo si y solo si es un lenguaje de NP y cualquier otro lenguaje de NP se reduce a él.

De manera informal, podemos decir que los problemas NP-completos son los más difíciles de resolver dentro de la clase NP. No se espera que puedan resolverse en tiempo polinomial por una MT determinista. En consecuencia, son candidatos perfectos para basar la seguridad de un criptosistema en ellos.

1.2. Álgebra

Comenzamos presentando la estructura algebraica básica para definir los códigos.

Definición 1.2.1 (Cuerpo)**.** [yVP03] Un cuerpo es una 5-tupla $(\mathbb{F}, +, \cdot, 0, 1)$, con $0, 1 \in \mathbb{F}$ verificando:

1. $(\mathbb{F}, +)$ es un grupo abeliano, donde a la operación + la llamamos suma y el elemento neutro es 0.
2. $(\mathbb{F}^* = \mathbb{F} \setminus \{0\}, \cdot)$ es también un grupo abeliano, donde a la operación $\cdot$ la llamamos multiplicación y el elemento neutro es 1.
3. $\forall a, b, c \in \mathbb{F}, a \cdot (b + c) = a \cdot b + a \cdot c$ (Distributividad de la multiplicación respecto a la suma).

Cuando $\mathbb{F}$ tenga un número finito de elementos q, lo llamaremos cuerpo finito, denotándolo como $\mathbb{F}_q$.

Observación 1.2.1. Un subconjunto de un cuerpo será un subcuerpo si tiene estructura de cuerpo al dotarlo de las operaciones de suma y multiplicación restringidas.

En el siguiente teorema resumimos las propiedades más importantes a la hora de empezar a trabajar con cuerpos finitos.

Teorema 1.2.1. *[yVP03] Sea $\mathbb{F}_q$ un cuerpo finito con q elementos. Se cumple:*

1. *$q = p^m$ para algún primo p.*
2. *$\mathbb{F}_q$ contiene el subcuerpo $\mathbb{F}_p$.*
3. *$\mathbb{F}_q$ es un espacio vectorial (lo definimos más adelante) sobre $\mathbb{F}_p$ de dimensión m.*
4. *$p\alpha = 0$ para todo $\alpha \in \mathbb{F}_q$, y.*
5. *$\mathbb{F}_q$ es único salvo isomorfismo.*

A raíz del punto 4 del teorema 1.2.1, se deriva el siguiente concepto.

Definición 1.2.2. (Característica) [yVP03] La característica de un cuerpo se define como el entero positivo más pequeño p verificando que el resultado de sumar 1 p veces es 0. Si no existe dicho entero, entonces la característica del cuerpo será 0. La característica será siempre un número primo.

Veamos los casos de cuerpos finitos más simples posibles, aunque a la vez los más comunes en el caso de los códigos que vamos a estudiar.

Ejemplo 1.2.1. [yVP03] El cuerpo binario $\mathbb{F}_2$ con dos elementos $\{0, 1\}$ tiene las siguientes tablas de suma y multiplicación:

$+$	0	1
0	0	1
1	1	0

$\cdot$	0	1
0	0	0
1	0	1

El cuerpo ternario $\mathbb{F}_3$ con tres elementos $\{0, 1, 2\}$ tiene las siguientes tablas de suma y multiplicación módulo 3:

$+$	0	1	2
0	0	1	2
1	1	2	0
2	2	0	1

$\cdot$	0	1	2
0	0	0	0
1	0	1	2
2	0	2	1

El cuerpo cuaternario $\mathbb{F}_4$ con cuatro elementos $\{0, 1, \omega, \overline{\omega}\}$ es más complicado. Tiene las siguientes tablas de suma y multiplicación:

$+$	0	1	ω	$\overline{\omega}$
0	0	1	ω	$\overline{\omega}$
1	1	0	$\overline{\omega}$	ω
ω	ω	$\overline{\omega}$	0	1
$\overline{\omega}$	$\overline{\omega}$	ω	1	0

$\cdot$	0	1	ω	$\overline{\omega}$
0	0	0	0	0
1	0	1	ω	$\overline{\omega}$
ω	0	ω	$\overline{\omega}$	1
$\overline{\omega}$	0	$\overline{\omega}$	1	ω

Veamos ahora algunos casos especiales de anillos.

Definición 1.2.3. (Dominio de Integridad)[yVP03] Un anillo conmutativo será un dominio de integridad si el producto de dos elementos cualesquiera diferentes de 0 es también distinto de 0.

Aunque para anillos existe también el concepto de subanillo, que es el análogo de los subgrupos y subcuerpos en el caso de los grupos y cuerpos respectivamente, los subconjuntos de los anillos más utilizados son los siguientes.

Definición 1.2.4. (Ideal) Sea R un anillo conmutativo. Un subconjunto $I \subseteq R, I \neq \emptyset$ será un ideal si verifica que, $\forall a, b \in I, a - b \in I$, y $\forall a \in I, r \in R, ar \in I$. El ideal $\mathcal{I}$ será principal si existe un $a \in R$ tal que $\mathcal{I} = \{ra | r \in R\}$; este ideal será denotado por (a).

Juntando los conceptos ya definidos de dominio de integridad e ideal principal, llegamos a la siguiente definición.

Definición 1.2.5. (Dominio de Ideales Principales) Un dominio de ideales principales es un dominio de integridad en el cual todos los ideales son principales.

Un ejemplo de dominio de ideales principales que va a aparecer reiteradamente a lo largo del trabajo es el de los anillos de polinomios sobre un cuerpo finito.

Ejemplo 1.2.2. [yVP03] Sea x una indeterminada. El conjunto de los polinomios en x con coeficientes en $\mathbb{F}_q$ es denotado por $\mathbb{F}_q[x]$. Usando la suma y multiplicación de polinomios usuales, podemos definir un anillo conmutativo sobre $\mathbb{F}_q[x]$. De hecho, es un dominio de integridad. Dado un polinomio $f(x) = \sum_{i=0}^{n} a_i x^i \in \mathbb{F}_q[x]$, a_i es el coeficiente del término $a_i x^i$ de grado i. El grado de un polinomio $f(x)$ es el mayor grado de cualquier término con coeficiente distinto de 0 (dicho coeficiente será llamado líder) y es denotado por gr $f(x)$; el polinomio cero no tiene grado.

Veamos la definición de irreducible para el anillo $\mathbb{F}_q[x]$, que es el que más nos va a interesar en nuestro caso.

Definición 1.2.6. (Polinomio Irreducible) [yVP03] Un polinomio no constante $f(x) \in \mathbb{F}_q[x]$ es irreducible sobre $\mathbb{F}_q$ si $\nexists g(x), h(x) \in \mathbb{F}_q[x]$ con
gr $g(x)$, gr $h(x) <$ gr $f(x) : f(x) = g(x)h(x)$.

Pasemos ahora con algunas definiciones previas a la presentación de dos resultados importantes en el estudio de los anillos de polinomios.

Definición 1.2.7. (Polinomio Mónico) Un polinomio es mónico si su coeficiente líder es 1.

Definición 1.2.8. (Divisibilidad en $\mathbb{F}_q[x]$) [yVP03] Sean $f(x)$ y $g(x)$ dos polinomios en $\mathbb{F}_q[x]$. Diremos que $f(x)$ divide a $g(x)$, denotado como $f(x)|g(x)$, si existe un polinomio $h(x) \in \mathbb{F}_q[x]$ tal que $g(x) = f(x)h(x)$. El polinomio $f(x)$ es llamado un divisor o factor de $g(x)$.

Definición 1.2.9. (Máximo Común Divisor) [yVP03] El máximo común divisor de $f(x)$ y $g(x)$, con $f(x) \neq 0$ o $g(x) \neq 0$, es el polinomio mónico $h(x) \in \mathbb{F}_q[x]$ verificando que $\text{gr } h(x) \geq \text{gr } q(x), \forall q(x) \in \mathbb{F}_q[x]$ y $h(x)|f(x), h(x)|g(x)$. El máximo común divisor de dos polinomios está únicamente determinado, y se denota como $mcd(f(x), g(x))$. Los polinomios $f(x)$ y $g(x)$ son primos relativos si $mcd(f(x), g(x)) = 1$.

La buena noticia es que la operación de división, con su correspondiente obtención de cociente y resto, que hacíamos en los números enteros puede ser trasladada fácilmente a los anillos de polinomios. De ello trata precisamente el siguiente resultado.

Teorema 1.2.2. *(Algoritmo de la División) [yVP03] Sean $f(x), g(x) \in \mathbb{F}_q[x]$ con $g(x) \neq 0$.*

1. *(Algoritmo de la División) Existen únicamente dos polinomios $h(x), r(x) \in \mathbb{F}_q[x]$ tal que*

$$f(x) = g(x)h(x) + r(x)$$

   donde gr $r(x) <$ gr $g(x)$ o $r(x) = 0$

2. *Si $f(x) = g(x)h(x) + r(x)$, entonces $mcd(f(x), g(x)) = mcd(g(x), r(x))$.*

Podemos usar el algoritmo de la división recursivamente junto con la parte 2 del teorema 1.2.2 para obtener el mcd de dos polinomios. Este proceso es conocido como el algoritmo de Euclides, que es presentado en el siguiente teorema.

Teorema 1.2.3. *[yVP03] Sean $f(x)$ y $g(x)$ polinomios en $\mathbb{F}_q[x]$ con $g(x)$ distinto de 0.*

1. *Llevamos a cabo la siguiente secuencia de pasos hasta que $r_n(x) = 0$ para algún n:*

$$\begin{aligned}
f(x) &= g(x)h_1(x) + r_1(x), && \text{donde gr } r_1(x) < \text{gr } g(x),\\
g(x) &= r_1(x)h_2(x) + r_2(x), && \text{donde gr } r_2(x) < \text{gr } r_1(x),\\
r_1(x) &= r_2(x)h_3(x) + r_3(x), && \text{donde gr } r_3(x) < \text{gr } r_2(x),\\
&\vdots\\
r_{n-3}(x) &= r_{n-2}(x)h_{n-1}(x) + r_{n-1}(x), && \text{donde gr } r_{n-1}(x) < \text{gr } r_{n-2}(x),\\
r_{n-2}(x) &= r_{n-1}(x)h_n(x) + r_n(x), && \text{donde } r_n(x) = 0.
\end{aligned}$$

   Entonces mcd(f(x),g(x)) = $cr_{n-1}(x)$, donde $c \in \mathbb{F}_q$ es escogido de manera que $cr_{n-1}(x)$ sea mónico.

2. *Existen polinomios $a(x), b(x) \in \mathbb{F}_q[x]$ tal que*

$$a(x)f(x) + b(x)g(x) = mcd(f(x), g(x)).$$

Observación 1.2.2. [yVP03] Si queremos construir un cuerpo de característica p, comenzamos con un polinomio $f(x) \in \mathbb{F}_p[x]$ que es irreducible sobre $\mathbb{F}_p$. Supongamos que $f(x)$ tiene grado m. Usando el algoritmo de Euclides, podemos probar que el anillo cociente $\frac{\mathbb{F}_p[x]}{(f(x))}$ es un cuerpo y por lo tanto un cuerpo finito $\mathbb{F}_q$ con $q = p^m$ elementos. Cada elemento de este anillo cociente es una clase lateral $g(x) + (f(x))$, donde $g(x)$ está determinado de forma única si tiene grado igual o inferior a $m-1$. Podemos simplificar la notación escribiendo la clase lateral como un vector de $\mathbb{F}_p^m$ usando la correspondencia
$g_{m-1}x^{m-1} + g_{m-2}x^{m-2} + ... + g_1x + g_0 + (f(x)) \longleftrightarrow g_{m-1}g_{m-2}\cdots g_1g_0$.
Veamos otra versión del algoritmo de Euclides que está orientada a la obtención de los denominados coeficientes de Bézout, y que utilizaremos cuando presentemos los códigos Reed-Solomon generalizados.

Teorema 1.2.4. *[Halb] Usando la misma notación que en el Teorema anterior, supongamos que gr $f(x) \geq$ gr $g(x)$ con $f(x) \neq 0$. En el paso i construimos la ecuación*

$$E_i : r_i(x) = s_i(x)f(x) + t_i(x)g(x).$$

La ecuación E_i es construida a partir de E_{i-1} y E_{i-2}, siendo la inicialización apropiada:

$$\begin{array}{lll} r_{-1}(x) = f(x); & s_{-1}(x) = 1; & t_{-1}(x) = 0; \\ r_0(x) = g(x); & s_0(x) = 0; & t_0(x) = 1 \end{array}$$

Paso i. Empezando con $r_{i-2}(x)$ y $r_{i-1}(x)$ $(\neq 0)$ usamos el algoritmo de la división para definir $h_i(x)$ y $r_i(x)$:

$$r_{i-2}(x) = h_i(x)r_{i-1}(x) + r_i(x) \text{ con gr } r_i(x) < \text{ gr } r_{i-1}(x).$$

A continuación definimos $s_i(x)$ y $t_i(x)$ por:

$$s_i(x) = s_{i-2}(x) - h_i(x)s_{i-1}(x);$$

$$t_i(x) = t_{i-2}(x) - h_i(x)t_{i-1}(x).$$

Entonces nos queda la ecuación

$$E_i : r_i(x) = s_i(x)f(x) + t_i(x)g(x).$$

Comenzamos con $i = 0$. Si tenemos que $r_i(x) \neq 0$, entonces continuamos al paso $i+1$. En algún momento habrá un i con $r_i(x) = 0$. En ese momento paramos y declaramos el mcd$(f(x), g(x))$ como el único múltiplo mónico del polinomio no nulo $r_{i-1}(x)$.

Cuando trabajamos con un cuerpo finito, es necesario que podamos multiplicar de la manera más sencilla posible. En primer lugar, presentamos el siguiente teorema.

Teorema 1.2.5. *[yVP03] Se verifica:*

1. *El grupo $\mathbb{F}_q^*$ es cíclico de orden $q-1$ bajo la multiplicación en $\mathbb{F}_q$.*
2. *Si γ es un generador de este grupo cíclico, entonces*

$$\mathbb{F}_q = \{0, 1 = \gamma^0, \gamma, \gamma^2, ..., \gamma^{q-2}\}$$

y $\gamma^i = 1$ si y solo si $(q-1)|i$.

Cada generador γ de $\mathbb{F}_q^*$ es llamado un elemento primitivo de $\mathbb{F}_q$. Cuando los elementos distintos de 0 de un cuerpo finito son expresados como potencias de γ, la multiplicación en el cuerpo se puede llevar a cabo fácilmente usando la regla $\gamma^i\gamma^j = \gamma^{i+j} = \gamma^s$, donde $0 \leq s \leq q-2$ e $i+j \equiv s$ (mod q - 1).

Sea γ un elemento primitivo de $\mathbb{F}_q$. Entonces $\gamma^{q-1} = 1$ por definición. Por lo tanto $(\gamma^i)^{q-1} = 1$ para $0 \leq i \leq q-2$, demostrando que los elementos de $\mathbb{F}_q^*$ son raíces de $x^{q-1} - 1 \in \mathbb{F}_p[x]$ y, en consecuencia, de $x^q - x$. Como 0 es una raíz de $x^q - x$, y un polinomio de grado q sobre $\mathbb{F}_q$ puede tener como mucho q raíces en alguna extensión de $\mathbb{F}_q$, tenemos que los elementos de $\mathbb{F}_q$ son precisamente las raíces de $x^q - x$, obteniendo el siguiente teorema.

Teorema 1.2.6. *[yVP03] Los elementos de $\mathbb{F}_q$ son precisamente las raíces de $x^q - x$.*

En el teorema 1.2.1 afirmamos que el cuerpo con $q = p^m$ elementos es único. El teorema 1.2.6 muestra que dicho cuerpo es el más pequeño conteniendo $\mathbb{F}_p$ y todas las raíces de $x^q - x$. Dicho cuerpo es llamado cuerpo de descomposición del polinomio $x^q - x$ sobre $\mathbb{F}_p$, esto es, la extensión de cuerpos más pequeña de $\mathbb{F}_p$ conteniendo todas las raíces del polinomio. En general, los cuerpos de descomposición de un polinomio fijo sobre un cuerpo son isomorfos.

Sea $\mathbb{E}$ una extensión finita de $\mathbb{F}_q$. Entonces $\mathbb{E}$ es un espacio vectorial sobre $\mathbb{F}_q$ y en consecuencia $\mathbb{E} = \mathbb{F}_{q^t}$ para algún entero positivo t. Por el teorema 1.2.6, cada elemento α de $\mathbb{E}$ es una raíz del polinomio $x^{q^t} - x$. Surge, por lo tanto, el siguiente concepto.

Definición 1.2.10. (Polinomio Mínimo) [yVP03] Sea $\alpha \in \mathbb{F}_{q^t}$. Llamaremos polinomio mínimo de α en $\mathbb{F}_q$, $M_\alpha \in \mathbb{F}_q[x]$, al polinomio mónico de grado más pequeño que tenga a α como raíz.

En el siguiente teorema presentamos algunos hechos básicos acerca de los polinomios mínimos.

Teorema 1.2.7. *[yVP03] Sea $\mathbb{F}_{q^t}$ una extensión de $\mathbb{F}_q$ y sea α un elemento de $\mathbb{F}_{q^t}$ con polinomio mínimo $M_\alpha(x)$ en $\mathbb{F}_q[x]$. Tenemos que:*

1. *$M_\alpha(x)$ es irreducible sobre $\mathbb{F}_q$.*
2. *Si $g(x)$ es un polinomio en $\mathbb{F}_q[x]$ satisfaciendo $g(\alpha) = 0$, entonces $M_\alpha(x) | g(x)$.*
3. *$M_\alpha(x)$ es única; esto es, existe solo un polinomio mónico en $\mathbb{F}_q[x]$ de grado mínimo que tiene a α como raíz.*

Si comenzamos con un polinomio irreducible $f(x)$ sobre $\mathbb{F}_q$ de grado r, podemos añadir una raíz de $f(x)$ a $\mathbb{F}_q$ para obtener el cuerpo $\mathbb{F}_{q^r}$. De hecho, todas las raíces de $f(x)$ están en $\mathbb{F}_{q^r}$.

Teorema 1.2.8. *[yVP03] Sea $f(x)$ un polinomio mónico irreducible sobre $\mathbb{F}_q$ de grado r. Entonces:*

1. *Todas las raíces de $f(x)$ están en $\mathbb{F}_{q^r}$ y en cualquier cuerpo conteniendo a $\mathbb{F}_q$ junto con una raíz de $f(x)$.*
2. *$f(x) = \prod_{i=1}^{r}(x - \alpha_i)$, donde $\alpha_i \in \mathbb{F}_{q^r}$ para $1 \leq i \leq r$.*
3. *$f(x) | x^{q^r} - x$.*

El teorema 1.2.8 es también cierto para los polinomios mínimos $M_\alpha(x)$ sobre $\mathbb{F}_q$ ya que, en particular, son mónicos irreducibles.

Teorema 1.2.9. *[yVP03] Sea $\mathbb{F}_{q^t}$ una extensión de $\mathbb{F}_q$ y α un elemento de $\mathbb{F}_{q^t}$ con polinomio mínimo $M_\alpha(x)$ en $\mathbb{F}_q[x]$. Se verifica lo siguiente:*

1. *$M_\alpha(x) | x^{q^t} - x$.*
2. *$M_\alpha(x)$ tiene raíces distintas, todas ellas en $\mathbb{F}_{q^t}$.*
3. *El grado de $M_\alpha(x)$ divide a t.*
4. *$x^{q^t} - x = \prod_\alpha M_\alpha(x)$, donde α está indexado sobre algún subconjunto de $\mathbb{F}_{q^t}$ que enumera a los polinomios mínimos de todos los elementos de $\mathbb{F}_{q^t}$ exactamente una vez.*
5. *$x^{q^t} - x = \prod_f f(x)$, donde f está indexado sobre el conjunto de todos los polinomios mónicos irreducibles sobre $\mathbb{F}_q$ cuyo grado divide a t.*

Como veremos más adelante cuando nos adentremos en el estudio de los códigos cíclicos, el problema central va a girar en torno a la factorización de los polinomios de la forma $x^n - 1$ sobre un cuerpo finito $\mathbb{F}_q$. Veamos un ejemplo sencillo de ello usando los resultados ya introducidos.

Ejemplo 1.2.3. Sea $x^4 - 1 \in \mathbb{F}_2[x]$. Los elementos de $\mathbb{F}_4$ son $\{0, 1, \omega, \omega + 1\}$, donde ω denota una de las raíces del polinomio irreducible $x^2 + x + 1 \in \mathbb{F}_2[x]$, que es el que usamos para construir $\mathbb{F}_4$. x es claramente el polinomio irreducible de 0 sobre $\mathbb{F}_2[x]$, así como $x - 1$ de 1. Como $\omega + 1$ también es raíz de $x^2 + x + 1$, y cada polinomio irreducible se incluye una sola vez por el teorema anterior, tenemos que $x^4 - 1$ factoriza como $x^4 - 1 = x(x - 1)(x^2 + x + 1)$.

Los polinomios mínimos nos van a permitir establecer una relación de equivalencia entre los distintos elementos de una extensión. Dicha relación es conocida como relación de conjugación.

Definición 1.2.11. (Conjugado)[yVP03] Dos elementos de $\mathbb{F}_{q^t}$ que tengan el mismo polinomio mínimo sobre $\mathbb{F}_q[x]$ son llamados conjugados sobre $\mathbb{F}_q$.

Será importante encontrar todos los conjugados de $\alpha \in \mathbb{F}_q$, es decir, todas las raíces de $M_\alpha(x)$. Sabemos por el apartado 2 del teorema 1.2.9 que las raíces de $M_\alpha(x)$ son distintas y están en $\mathbb{F}_{q^t}$. Podemos encontrar estas raíces con la ayuda del siguiente teorema.

Teorema 1.2.10. *[yVP03] Sea $f(x)$ un polinomio en $\mathbb{F}_q[x]$ y α una raíz de $f(x)$ en alguna extensión $\mathbb{F}_{q^t}$. Entonces:*

1. *$f(x^q) = f(x)^q$.*
2. *α^q es también una raíz de $f(x)$ en $\mathbb{F}_q$.*

Aplicando reiteradamente este teorema observamos que $\alpha, \alpha^q, \alpha^{q^2}, \ldots$ son todas raíces de $M_\alpha(x)$. ¿Dónde termina esta secuencia? Parará después de r términos, donde $\alpha^{q^r} = \alpha$. Supongamos ahora que γ es un elemento primitivo de $\mathbb{F}_{q^t}$. Se cumple que $\alpha = \gamma^s$ para algún s. Entonces $\alpha^{q^r} = \alpha$ si y solo si $\gamma^{sq^r - s} = 1$. Por el Teorema 1.2.5, $sq^r \equiv s \pmod{q^t - 1}$. Basado en esto, presentamos el siguiente concepto.

Definición 1.2.12. [yVPo3] Definimos la clase lateral q-ciclotómica de s módulo $q^t - 1$ como el conjunto $C_s = \{s, sq, ..., sq^{r-1}\}$ (mod $q^t - 1$), donde r es el entero positivo más pequeño tal que $sq^r \equiv s$ (mod $q^t - 1$).

Observación 1.2.3. Los conjuntos C_s particionan $\{0, 1, 2, ..., q^t - 2\}$ en subconjuntos disjuntos. Cuando listamos las clases laterales ciclotómicas, es común listar C_s solo una vez, siendo s el elemento más pequeño de la clase lateral.

Ahora sabemos que las raíces de $M_\alpha(x) = M_{\gamma^s}(x)$ incluyen a $\{\gamma^i | i \in C_s\}$. De hecho éstas son todas las raíces. Por lo tanto, si conocemos el tamaño de C_s sabemos también el grado de $M_{\gamma^s}(x)$, ya que coinciden.

Teorema 1.2.11. *[yVPo3] Si γ es un elemento primitivo de $\mathbb{F}_{q^t}$ entonces el polinomio mínimo de γ^s sobre $\mathbb{F}_q$ es*

$$M_{\gamma^s}(x) = \prod_{i \in C_s} (x - \gamma^i).$$

Para analizar la estructura de un cuerpo, será útil saber el número de elementos primitivos en $\mathbb{F}_q$ y cómo encontrarlos todos una vez conocido uno de ellos. Como $\mathbb{F}_q^*$ es cíclico, recordamos una serie de hechos acerca de los grupos cíclicos finitos. En cualquier grupo cíclico finito G de orden n con generador g, los generadores de G son precisamente los elementos g^i donde $mcd(i, n) = 1$. Sea $\phi(n)$ el número de enteros i con $1 \leq i \leq n$ tal que $mcd(i, n) = 1$; ϕ es llamada la función ϕ de Euler. Por lo tanto hay $\phi(n)$ generadores de G. El orden de un elemento $\alpha \in G$ es el entero positivo más pequeño i tal que $\alpha^i = 1$. Un elemento de G tiene orden d si y solo si $d|n$. Además g^i tiene orden $d = \frac{n}{mcd(i,n)}$ y hay $\phi(d)$ elementos de orden d. Cuando hablamos de orden de un elemento $\alpha \in \mathbb{F}_q$, que además verifica $\alpha \in \mathbb{F}_q^*$, el orden de α es su orden en el grupo multiplicativo $\mathbb{F}_q^*$. En particular, los elementos primitivos de $\mathbb{F}_q$ son aquellos de orden $q - 1$. Estamos ya en condiciones de presentar el siguiente teorema.

Teorema 1.2.12. *[yVPo3] Sea γ un elemento primitivo de $\mathbb{F}_q$.*

1. *Existen $\phi(q-1)$ elementos primitivos en $\mathbb{F}_q$; éstos son los elementos γ^i donde $mcd(i, q-1) = 1$.*
2. *Para cualquier d donde $d|(q-1)$, hay $\phi(d)$ elementos en $\mathbb{F}_q$ de orden d; éstos son los elementos $\phi^{\frac{(q-1)i}{d}}$ donde $mcd(i, d) = 1$.*

A continuación presentamos un subgrupo muy importante dentro del grupo multiplicativo de cualquier cuerpo finito.

Definición 1.2.13. (raíz de la unidad n-ésima)[yVPo3] Un elemento $\xi \in \mathbb{F}_q$ es una raíz de la unidad n-ésima si $\xi^n = 1$, y será primitiva si además cumple $\xi^s \neq 1$ para $0 < s < n$.

Observación 1.2.4. Un elemento primitivo γ de $\mathbb{F}_q$ es por lo tanto una raíz primitiva de la unidad (q-1)-ésima. Del teorema 1.2.12 se deduce que el cuerpo $\mathbb{F}_q$ contendrá una raíz primitiva de la unidad n-ésima si y solo si $n|(q-1)$, en cuyo caso $\gamma^{\frac{q-1}{n}}$ será tal raíz.

Ahora que ya estamos familiarizados con el uso de polinomios, presentemos un concepto relacionado que nos hará falta más adelante.

Definición 1.2.14 (Base de Lagrange)**.** [Wik] Dado un conjunto de $k+1$ nodos $\{x_0, x_1, ..., x_k\}$, los cuales deben ser todos diferentes, es decir, $x_j \neq x_m$ para índices $j \neq m$, la base de Lagrange para polinomios de grado $\leq k$ para esos nodos es el conjunto de polinomios $\{l_o(x), l_1(x), ..., l_k(x)\}$, cada uno de grado k, los cuales toman valores $l_j(x_m) = 0$ si $m \neq j$ y $l_j(x_j) = 1$. Usando la delta de Kronecker esto puede ser escrito como $l_j(x_m) = \delta_{jm}$. Cada polinomio de base puede ser explícitamente descrito por el producto:

$$l_j(x) = \frac{x - x_0}{x_j - x_0} \cdots \frac{x - x_{j-1}}{x_j - x_{j-1}} \frac{x - x_{j+1}}{x_j - x_{j+1}} \cdots \frac{x - x_k}{x_j - x_k} = \prod_{\substack{0 \leq m \leq k \\ m \neq j}} \frac{x - x_m}{x_j - x_m}.$$

Cabe mencionar que el numerador $\prod_{m \neq j}(x - x_m)$ tiene k raíces en los nodos $\{x_m\}_{m \neq j}$ mientras el denominador $\prod_{m \neq j}(x_j - x_m)$ escala el polinomio resultante de manera que $l_j(x_j) = 1$.

Definición 1.2.15 (Polinomio de Interpolación de Lagrange)**.** [Wik] El polinomio de interpolación de Lagrange para los nodos de la definición 1.2.14 y sus valores correspondientes $\{y_0, y_1, ..., y_k\}$ es la combinación lineal $L(x) = \sum_{j=0}^{k} y_j l_j(x)$.

Cada polinomio de base tiene grado k, por lo que la suma $L(x)$ tiene grado $\leq k$ e interpola los datos porque $L(x_m) = \sum_{j=0}^{k} y_j l_j(x_m) = \sum_{j=0}^{k} y_j \delta_{jm} = y_m$.

2 Códigos Lineales

En este capítulo comenzamos nuestro estudio de la teoría de códigos, el primer pilar sobre el que va a sostenerse nuestro trabajo. Introduciremos el concepto de código lineal para pasar después a la presentación de alguna de sus propiedades más importantes, y culminaremos con la demostración de la NP-Completitud del problema de la decodificación de un código lineal binario, que nos será de gran utilidad en la parte de criptografía. El capítulo se basa en los contenidos de [yVP03] y [ERB78].

2.1. ¿Qué es un código lineal?

Para comenzar nuestro estudio, a lo largo de esta sección realizaremos un breve recorrido por los puntos más importantes de la Teoría de Códigos, un área de estudio originada por Hamming en su intento de arreglar los errores de cálculo en un ordenador. La problemática básica de la que parte es que la información puede verse alterada debido a la interacción con el canal de comunicación por el que viaja, es decir, lo que el receptor recibe puede ser diferente a lo que fue enviado. Por ello, la teoría de códigos pretende proveer herramientas para determinar qué mensaje fue enviado en base a lo que se recibió. Mientras que Hamming fue uno de los primeros investigadores en desarrollar métodos prácticos, quien proporcionó el marco teórico fue un contemporáneo suyo, Shannon. En su estudio, Shannon identificó un número llamado la capacidad del canal y demostró que se puede lograr una comunicación tan veraz como se quiera a cualquier tasa de información enviada, siempre que se mantenga por debajo de dicha capacidad.

Por ejemplo, en la transmisión de imágenes de planetas desde el espacio exterior es muy poco eficiente tener que reenviar alguna de estas imágenes. Por lo tanto, si parte de la información enviada se ve alterada debido a la adición de ruido (a causa de, por ejemplo, perturbaciones térmicas) durante la transmisión, las imágenes recibidas en la Tierra pueden resultar de muy poca utilidad. Los resultados de Shannon garantizan que la información pueda ser codificada antes de la transmisión, por lo que la información alterada puede ser decodificada al nivel de veracidad especificado.

Un canal de comunicación está formado por 5 elementos principales. En primer lugar, una fuente, desde donde un mensaje x va a ser enviado. Si ninguna modificación es hecha y el mensaje es transmitido directamente sobre el canal, cualquier ruido perturbará el mensaje y no podremos recuperar el original. Por lo tanto, en el codificador añadiremos algo de redundancia, obteniendo una palabra de código c, que será la que va a ser enviada por el canal. El potencial ruido añadido por el canal lo denotaremos mediante el vector e, que es añadido a c, de manera que lo que llega al decodificador es el vector $y = c + e$. A partir de y, el decodificador producirá una estimación de x, denotada como $\hat{x}$.

Ejemplo 2.1.1. Retomando el ejemplo de la transmisión de imágenes de planetas, en este caso la fuente sería el satélite, el canal el espacio exterior junto con el hardware utilizado

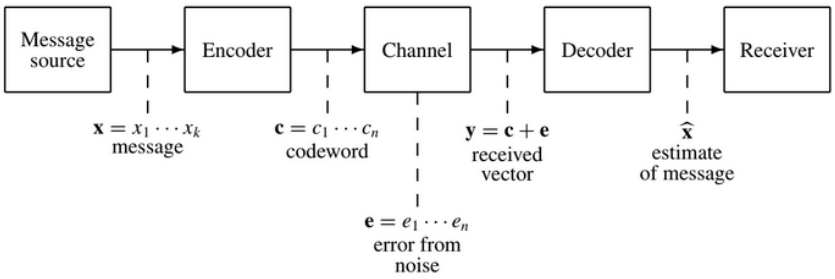

Figura 2.1: Canal de comunicación [yVP03]

para enviar y recibir la información, y el receptor sería la estación en la Tierra (también podría darse la comunicación en el otro sentido, alternándose los roles).

De entre todos los tipos de códigos, los lineales son los más estudiados. Debido a su estructura algebraica, son más fáciles de describir, codificar y decodificar que los no lineales. En este trabajo, el alfabeto para los códigos lineales será un cuerpo finito.

Definición 2.1.1 (Código Lineal)**.** Sea $\mathbb{F}_q^n$ el espacio vectorial de todas las n-tuplas sobre el cuerpo finito $\mathbb{F}_q$. Un (n, M)-código C sobre $\mathbb{F}_q$ es un subconjunto de $\mathbb{F}_q^n$ de tamaño M. Normalmente escribimos los vectores $(a_1, a_2, ..., a_n)$ en $\mathbb{F}_q^n$ en la forma $a_1a_2...a_n$ y llamaremos a los vectores en C palabras código. Si C es un subespacio vectorial k-dimensional de $\mathbb{F}_q^n$, entonces diremos que C es un $[n, k]_q$-código lineal.

A todos los códigos sobre $\mathbb{F}_2$ los llamaremos binarios. Similarmente, a aquellos sobre $\mathbb{F}_3$ los llamaremos ternarios, y sobre $\mathbb{F}_4$ cuaternarios.

Observación 2.1.1. Si C es un código no lineal (o que se desconozca su linearidad), un subcódigo de C es cualquier subconjunto de C. Si C es lineal, un subcódigo será un subconjunto de C que debe ser también lineal, es decir, un subespacio vectorial suyo.

2.2. Propiedades de los códigos lineales

Vamos a estudiar ahora todas las propiedades de los códigos lineales que vamos a necesitar a lo largo del trabajo. Las dos maneras más comunes de presentar un código lineal es, o bien con la matriz generadora, o bien con la matriz de paridad. Comenzamos por la definición de la primera.

Definición 2.2.1 (Matriz generadora)**.** Una matriz generadora G para un $[n, k]$-código C es cualquier matriz de tamaño $k \times n$ cuyas filas formen una base para C.

Para cualquier conjunto de k columnas independientes de una matriz generadora G, el conjunto de coordenadas correspondientes forma un conjunto de información para C. Las restantes $r = n - k$ coordenadas son conocidas como conjunto de redundancia, siendo r la redundancia de C. Si las primeras k coordenadas forman un conjunto de información, el código tiene una matriz generadora única de la forma $[I_k|A]$, donde I_k es la matriz identidad $k \times k$. Una tal matriz generadora diremos que está en forma estándar. Vamos ahora con el concepto de matriz de paridad.

Definición 2.2.2 (Matriz de paridad)**.** Una matriz de paridad H para un $[n,k]$-código C es una matriz de tamaño $(n-k)\times n$ que verifica $C=\{x\in\mathbb{F}_q^n | Hx^T=0\}$.

En general hay múltiples matrices generadoras y de paridad para un mismo código.

Observación 2.2.1. La aplicación $x\to Hx^T$ es una transformación lineal de $\mathbb{F}_q^n$ a $\mathbb{F}_q^{n-k}$ cuyo núcleo es C. Sabemos que la dimensión de C como subespacio de $\mathbb{F}_q^n$ es k, luego tenemos que la dimensión de la imagen de esta aplicación es $n-k$ y por tanto, el rango de H también.

Ejemplo 2.2.1. La matriz $G=[I_4|A]$,

$$\left(\begin{array}{cccc|ccc} 1&0&0&0&0&1&1\\ 0&1&0&0&1&0&1\\ 0&0&1&0&1&1&0\\ 0&0&0&1&1&1&1 \end{array}\right)$$

es una matriz generadora para un $[7,4]$-código binario que denotamos como $\mathcal{H}_3$. Una matriz de paridad para $\mathcal{H}_3$ sería

$$H=[A^T|I_3]=\left(\begin{array}{cccc|ccc} 0&1&1&1&1&0&0\\ 1&0&1&1&0&1&0\\ 1&1&0&1&0&0&1 \end{array}\right).$$

Este código es llamado $[7,4]$-código de Hamming.

La matriz generadora G de un $[n,k]$-código C es simplemente una matriz cuyas filas son independientes y generan el código. Las filas de la matriz de paridad H son independientes, por lo que H es la matriz generadora de algún código, llamado el dual u ortogonal de C y denotado como $C^\perp$. Cabe mencionar que $C^\perp$ es un $[n,n-k]$-código. Una manera alternativa de definir el código dual es usando productos escalares. Recordemos que el producto escalar usual de vectores $x=x_1...x_n, y=y_1...y_n$ en $\mathbb{F}_q^n$ es $x\cdot y=\sum_{i=1}^n x_iy_i$.

Definición 2.2.3 (Dual de un código)**.** Sea C un código lineal. Definimos su dual $C^\perp$ como $\{x\in\mathbb{F}_q^n | x\cdot c=0, \forall c\in C\}$.

Con la siguiente definición vamos a ver que además de tratar a $\mathbb{F}_q^n$ como espacio vectorial, lo vamos a ver como espacio métrico dotado de la siguiente distancia.

Definición 2.2.4. (Distancia de Hamming) La distancia (de Hamming) $d(x,y)$ entre dos vectores $x,y\in\mathbb{F}_q^n$ se define como el número de coordenadas en las que x e y difieren. La distancia (mínima) de un código C es la distancia más pequeña entre palabras código distintas.

La distancia de un código es importante a la hora de determinar su capacidad de corregir errores. Se cumple que, a mayor distancia mínima, mayor cantidad de errores pueden ser corregidos por el código.
Verifiquemos que la distancia de Hamming es efectivamente una distancia.

Teorema 2.2.1. *La función de distancia $d(x,y)$ satisface las cuatro propiedades siguientes:*

1. $d(x,y)\geq 0, \forall x,y\in\mathbb{F}_q^n$ *(no-negatividad).*
2. $d(x,y)=0\leftrightarrow x=y$.

3. $d(x,y) = d(y,x), \forall x,y \in \mathbb{F}_q^n$ *(simetría).*

4. $d(x,z) \leq d(x,y) + d(y,z), \forall x,y,z \in \mathbb{F}_q^n$ *(desigualdad triangular).*

Demostración. [Excb] Las tres primeras propiedades son triviales, por lo que vamos a centrarnos en la cuarta. Sea $x \in \mathbb{F}_q^n, x = x_1...x_n$. Para $x,y \in \mathbb{F}_q^n$ sea $D(x,y) = \{k : x_k \neq y_k\}$; por definición $d(x,y) = |D(x,y)|$, dónde $|\cdot|$ denota el cardinal del conjunto correspondiente. Supongamos ahora que $x,y,z \in \mathbb{F}_q^n$. Entonces

$$D(x,y) = D(x,z) \triangle D(z,y) \subseteq D(x,z) \cup D(z,y),$$

donde $\triangle$ denota la diferencia simétrica, es decir,
$D(x,z) \triangle D(z,y) = (D(x,z) \cup D(z,y)) \setminus (D(x,z) \cap D(z,y))$. Por lo tanto

$$d(x,y) = |D(x,y)| \leq |D(x,z) \cup D(z,y)| \leq |D(x,z)| + |D(z,y)| = d(x,z) + d(z,y).$$

□

Estrechamente ligado con la distancia recién definida está el siguiente concepto.

Definición 2.2.5 (Peso de Hamming)**.** El peso (de Hamming) $wt(x)$ de un vector $x \in \mathbb{F}_q^n$ es el número de coordenadas diferentes de cero en x.

El siguiente teorema relaciona peso y distancia.

Teorema 2.2.2. *Si $x,y \in \mathbb{F}_q^n$, entonces $d(x,y) = wt(x-y)$. Si C es un código lineal, la distancia mínima d coincide con el peso mínimo de las palabras código no nulas de C.*

Demostración. Sea w el peso mínimo de las palabras código no nulas de C. Si $u \in C$, entonces $wt(u) = d(u,0)$, luego $w \geq d$. Sean ahora $u,v \in C$ tales que $d(u,v) = d$, es decir, u y v tienen la misma distancia. Pero $d = d(u,v) = d(0,v-u) = wt(v-u) \geq w$, luego $w = d$. □

Este resultado nos va a permitir hablar indistintamente de peso mínimo y distancia mínima de un código. Si el peso mínimo d de un $[n,k]$-código es conocido, nos referiremos al código como $[n,k,d]$-código. Las dos definiciones que siguen componen un recorrido fugaz por la teoría de equivalencias entre códigos lineales.

Definición 2.2.6. (Matriz monomial) Una matriz monomial es una matriz cuadrada con exactamente una entrada distinta de cero en cada fila y columna.

Una vez conocido el concepto de matriz monomial, veamos su uso para definir un tipo de equivalencia entre códigos.

Definición 2.2.7 (Equivalencia Monomial)**.** Sean C_1 y C_2 códigos de la misma longitud sobre el cuerpo $\mathbb{F}_q$, y sea G_1 una matriz generadora para C_1. Entonces C_1 y C_2 son monomialmente equivalentes si existe una matriz monomial M tal que G_1M es una matriz generadora para C_2.

Una vez introducidos una serie de conceptos elementales, podemos comenzar a tratar los dos procesos fundamentales que intervienen en nuestro canal de comunicación, el de codificación y el de decodificación. Comenzamos por el de codificación. La forma más sencilla de codificar un mensaje es usando la matriz generadora. Sea C un $[n,k]$-código lineal sobre el cuerpo $\mathbb{F}_q$ con matriz generadora G. Dado un mensaje $x \in \mathbb{F}_q^k$, obtenemos la

palabra código $c = xG$. Si G está en forma estándar, las primeras k coordenadas de la palabra código c son los símbolos de información x; los restantes $n - k$ símbolos son los símbolos de redundancia, es decir, la redundancia añadida a x para así ayudar a recuperarlo en caso de que ocurran errores. La matriz G puede no estar en forma estándar. Sin embargo, si existen índices de columnas $i_1, i_2, ..., i_k$ tal que la matriz de tamaño $k \times k$ consistente de estas k columnas de G es la matriz identidad $k \times k$, entonces el mensaje es encontrado en las k coordenadas $i_1, i_2, ..., i_k$ de la palabra código, desordenadas pero intactas; esto es, el símbolo del mensaje x_j es la componente i_j de la palabra código. Si esto ocurre, diremos que el codificador es sistemático.

Ejemplo 2.2.2. Sea C el $[6, 3, 3]$-código binario con matriz generadora

$$G = \begin{pmatrix} 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \end{pmatrix}.$$

Supongamos que queremos codificar el mensaje $x = x_1x_2x_3$ para obtener la palabra código $c = c_1c_2...c_6$. Usando G para codificar tenemos que

$$c = xG = (x_1, x_2, x_3, x_1 + x_2, x_2 + x_3, x_1 + x_3).$$

Ya que hay una correspondencia uno a uno entre mensajes y palabras código, a menudo solo se trabaja con mensajes codificados (las palabras código) tanto en el envío como en la recepción. En ese caso, en el paso de decodificar en la Figura 2.1, estamos satisfechos con una estimación $\hat{c}$ obtenida del decodificador al recibir y, esperando que sea la palabra código c que fue transmitida. Sin embargo, si estamos interesados en el mensaje original, surge la pregunta de cómo recuperar el mensaje a partir de una palabra código. Si la palabra código es $c = xG$, y G está en forma estándar, el mensaje son las primeras k componentes de c; si el codificado es sistemático, es fácil recuperar el mensaje mirando en las coordenadas de G conteniendo la matriz identidad. ¿Qué podemos hacer en caso contrario? Como las filas de G son linealmente independientes, existe una matriz K de tamaño $n \times k$ tal que $GK = I_k$; K es llamada una inversa por la derecha de G y no es necesariamente única. Como $c = xG$, $cK = xGK = x$.

El proceso de decodificación, es decir, determinar qué palabra código (y en consecuencia qué mensaje x) fue enviada cuando un vector y es recibido, es más complejo que el de codificación. Comenzamos con un posible modelo matemático de un canal que transmite datos binarios para presentar el proceso de decodificación. Este modelo es llamado el canal binario simétrico (BSC de las siglas en inglés) con probabilidad de mezcla ϱ. Si 0 o 1 es enviado, la probabilidad de que sean recibidos sin error es $1 - \varrho$; si un 0 (respectivamente un 1) es enviado, la probabilidad de que un 1 (respectivamente un 0) sea recibido es ϱ. En la mayoría de casos prácticos ϱ es muy pequeño. Esto es un ejemplo de un canal discreto sin memoria (DMC de las siglas en inglés), un canal en el cual entradas y salidas son discretas y la probabilidad de error en un bit es independiente a la de los bits anteriores. Asumiremos que es más probable que un bit sea recibido correctamente que erróneamente, por lo que $\varrho < \frac{1}{2}$.
Si E_1 y E_2 son eventos, sea $prob(E_1)$ la probabilidad de que E_1 ocurra y $prob(E_1|E_2)$ la probabilidad de que E_1 ocurra supuesto que E_2 ha ocurrido. Supongamos que $c \in \mathbb{F}_2^n$ es enviado e $y \in \mathbb{F}_2^n$ es recibido y decodificado como $\hat{c} \in \mathbb{F}_2^n$. $Prob(y|c)$ es la probabilidad de

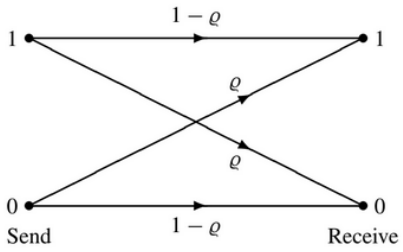

Figura 2.2: Esquema de un BSC [yVP03]

que y sea recibido supuesto que la palabra código c haya sido enviada. Esta probabilidad puede calcularse a partir de las estadísticas asociadas con el canal.
El decodificador podría por tanto escoger $\hat{c} = c$ para la palabra código c con $prob(y|c)$ máxima; tal decodificador es llamado de máxima verosimilitud (ML de las siglas en inglés). Simbólicamente, un decodificador ML calcula

$$\hat{c} = argmax_{c \in C} prob(y|c).$$

Consideremos ahora la decodificación ML sobre un BSC. Si $y = y_1...y_n$ y $c = c_1...c_n$,

$$prob(y|c) = \prod_{i=1}^{n} prob(y_i|c_i).$$

Dado que estamos en un DMC, y por lo tantos los errores en los bits son independientes. De acuerdo con Figura 2.2, $prob(y_i|c_i) = \varrho$ si $y_i \neq c_i$ y $prob(y_i|c_i) = 1 - \varrho$ si $y_i = c_i$. Por lo tanto

$$prob(y|c) = \varrho^{d(y,c)}(1-\varrho)^{n-d(y,c)} = (1-\varrho)(\frac{\varrho}{1-\varrho})^{d(y,c)}.$$

Ya que $0 < \varrho < \frac{1}{2}, 0 < \frac{\varrho}{1-\varrho} < 1$. En consecuencia, maximizar $prob(y|c)$ es equivalente a minimizar $d(y,c)$, es decir, encontrar la palabra código c más cercana al vector recibido y en términos de la distancia de Hamming; a este método se le conoce como decodificación del vecino más cercano. Por lo tanto, en un BSC, la decodificación máximo verosímil y la del vecino más cercano son equivalentes.
Sea $e = y - c$ de manera que $y = c + e$. El efecto del ruido en el canal de comunicación es la suma de un vector de error e a la palabra código c, y la meta de la decodificación es determinar e. La decodificación del vecino más cercano es equivalente a encontrar un vector e de peso mínimo tal que $y - e$ esté en el código. Este vector de error no es necesariamente único ya que puede haber más de una palabra código a distancia mínima de y.
Introduzcamos ahora un concepto útil a la hora de hablar de proximidad a una palabra código dada.

Definición 2.2.8 (Esfera)**.** La esfera de radio r centrada en un vector $u \in \mathbb{F}_q^n$ es definida como el conjunto $S_r(u) = \{v \in \mathbb{F}_q^n | d(u,v) \leq r\}$ de todos los vectores cuya distancia a u es menor o igual que r.

Estas esferas son disjuntas dos a dos si su radio es escogido suficientemente pequeño.

Teorema 2.2.3. *Si d es la distancia mínima de un código C y $t = [\frac{d-1}{2}]$, donde $[\cdot]$ denota la parte entera, entonces las esferas de radio t centradas en distintas palabras código son disjuntas.*

Demostración. Si $z \in S_t(c_1) \cap S_t(c_2)$, donde c_1 y c_2 son palabras código, entonces por la desigualdad triangular

$$d(c_1, c_2) \leq d(c_1, z) + d(z, c_2) \leq 2t < d.$$

Esto implica que $c_1 = c_2$. □

Corolario 2.2.1. *Con la notación del teorema anterior, si una palabra código c es enviada e y es recibida habiendo ocurrido t o menos errores, entonces c es la única palabra código más cercana a y. En particular, el método del vecino más cercano decodifica única y correctamente cualquier vector recibido en el cual, como mucho, hayan ocurrido t errores durante la transmisión.*

El problema ahora consiste en encontrar un algoritmo de decodificación eficiente que corrija hasta t errores. Uno de los algoritmos de decodificación más triviales es examinar todas las palabras código hasta que una sea encontrada a distancia t o menor del vector recibido. Sin embargo, éste es un algoritmo de decodificación realista solo para códigos con un número pequeño de palabras código. Otro algoritmo podría consistir en construir una tabla formada por todas las palabras código más cercanas para cada uno de los q^n vectores en $\mathbb{F}_q^n$ y luego consultar el vector recibido en la tabla para decodificarlo. Este método tampoco es viable si q^n es muy grande.
Sin embargo, para un $[n, k, d]$-código lineal C sobre $\mathbb{F}_q$ podemos construir un algoritmo usando una tabla con q^{n-k} en vez de q^n entradas, pudiendo encontrar la palabra código más cercana mirando una de las q^{n-k} entradas. Este método para códigos lineales es llamado decodificación mediante síndrome. Dado que nuestro código C es un subgrupo abeliano del grupo aditivo que forma $\mathbb{F}_q^n$, sus clases laterales $x + C$ dividen $\mathbb{F}_q^n$ en q^{n-k} conjuntos de tamaño q^k. Dos vectores x e y pertenecen a la misma clase lateral si y solo si $y - x \in C$. El peso de una clase lateral es el peso más pequeño de un vector en dicha clase, y cualquier vector con tal peso en la clase lateral es llamado un líder. Tenemos que cualquier clase lateral con peso inferior a $t = [(d-1)/2]$ tiene un único líder.
Vamos ahora con la definición más importante dentro de este método de decodificación.

Definición 2.2.9 (Síndrome)**.** Sea H una matriz de paridad para C. El síndrome de un vector $x \in \mathbb{F}_q^n$ con respecto a H es el vector en $\mathbb{F}_q^{n-k}$ definido como $syn(x) = Hx^T$.

El código C consiste en todos los vectores cuyo síndrome es 0, por la definición 2.2.2. Como H tiene rango $n - k$, cada vector en $\mathbb{F}_q^{n-k}$ será síndrome de algún vector.
El siguiente teorema nos pemite determinar si dos vectores están en la misma clase lateral usando el concepto recién definido.

Teorema 2.2.4. *Dos vectores pertenecen a la misma clase lateral si y solo si tienen el mismo síndrome.*

Demostración. Si $x_1, x_2 \in \mathbb{F}_q^n$ están en la misma clase lateral de C, entonces $x_1 - x_2 = c \in C$. Por lo tanto $syn(x_1) = H(x_2 + c)^T = Hx_2^T + Hc^T = Hx_2^T = syn(x_2)$. En consecuencia, x_1 y x_2 tienen el mismo síndrome. Por otro lado, si $syn(x_1) = syn(x_2)$, entonces $H(x_2 - x_1)^T = 0$ y por lo tanto $x_2 - x_1 \in C$. □

Denotaremos por C_s a la clase lateral de C formada por todos los vectores en $\mathbb{F}_q^n$ con síndrome s. Supongamos que una palabra código es enviada por un canal de comunicación y recibida como un vector y. En la decodificación por el vecino más cercano buscamos un vector e de peso mínimo tal que $y - e \in C$, siendo equivalente a encontrar un vector e de peso mínimo en la clase lateral conteniendo a y, esto es, un líder de dicha clase. El algoritmo de decodificación por síndrome es la siguiente implementación del método del vecino más cercano:

1. Para cada síndrome $s \in \mathbb{F}_q^{n-k}$, elegimos un líder e_s de la clase lateral C_s. Creamos una tabla enlazando el síndrome con el correspondiente líder.
2. Después de recibir el vector y, calculamos su síndrome s usando la matriz de paridad H.
3. y es decodificado como la palabra de código $y - e_s$.

A continuación vamos a presentar una cota para el número de palabras en un código lineal o no lineal, dadas la longitud n y la distancia mínima d de dicho código. Usaremos la notación $B_q(n,d)$ y $A_q(n,d)$ para referirnos al mayor número de palabras en un código lineal o arbitrario (lineal o no lineal), respectivamente, sobre $\mathbb{F}_q$ de longitud n y distancia mínima al menos d. Antes de presentar la cota, veamos una serie de propiedades elementales de $B_q(n,d)$ y $A_q(n,d)$.

Teorema 2.2.5. *$B_q(n,d) \leq A_q(n,d)$ y $B_q(n,d)$ es un entero no negativo potencia de q.*

Teorema 2.2.6. *$A_q(n,n) = B_q(n,n) = q$.*

Demostración. El código lineal de tamaño q formado por todos los múltiplos del vector formado por solo 1s, de longitud n, tiene distancia mínima d. Por el Teorema 2.2.5, $A_q(n,n) \geq B_q(n,n) \geq q$. Si $A_q(n,n) > q$, existe un código con más de q palabras y distancia mínima n. Por lo tanto, al menos dos de las palabras código coinciden en alguna coordenada, pero entonces estas dos palabras están a una distancia inferior a n, una contradicción. Tenemos finalmente que $A_q(n,n) = B_q(n,n) = q$. □

Teorema 2.2.7. *$A_q(n,d) \leq qA_q(n-1,d)$.*

Demostración. Sea C un código sobre $\mathbb{F}_q$ de longitud n y distancia mínima al menos d con $M = A_q(n,d)$ palabras código. Sea $C(\alpha)$ el subcódigo de C en el cual cada palabra código tiene α en la coordenada n. Entonces para algún α, $C(\alpha)$ contiene al menos $\frac{M}{q}$ palabras código. Eliminando la coordenada n de todas las palabras código producimos un código de longitud $n-1$ y distancia mínima d. Por lo tanto, $\frac{M}{q} \leq A_q(n-1,d)$ dándonos $A_q(n,d) \leq qA_q(n-1,d)$. □

La cota anunciada es la cota Singleton. Es una cota más bien débil, pero a raíz de ella surge la familia de códigos llamada códigos MDS, que contiene por ejemplo a los códigos Reed-Solomon que estudiaremos más adelante.

Teorema 2.2.8 (Cota Singleton)**.** *Para $d \leq n$*

$$A_q(n,d) \leq q^{n-d+1}.$$

Además, si un $[n,k,d]_q$-código lineal existe, entonces $k \leq n - d + 1$.

Demostración. La segunda afirmación se sigue de la primera por el Teorema 2.2.5 . Recordemos que $A_q(n,n) = q$ por el Teorema 2.2.6, dándonos la cota cuando $d = n$. Supongamos ahora que $d < n$. Por el Teorema 2.2.7 tenemos que $A_q(n,d) \leq qA_q(n-1,d)$. Inductivamente tenemos que $A_q(n,d) \leq q^{n-d}A_q(d,d)$. Como $A_q(d,d) = q$, concluimos finalmente que $A_q(n,d) \leq q^{n-d+1}$. □

Ejemplo 2.2.3. El $[6,3,4]$-código cuaternario $\mathcal{G}_6$ tiene matriz generadora G_6 dada por

$$G_6 = \left(\begin{array}{ccc|ccc} 1 & 0 & 0 & 1 & \omega & \omega \\ 0 & 1 & 0 & \omega & 1 & \omega \\ 0 & 0 & 1 & \omega & \omega & 1 \end{array}\right).$$

Es conocido como el código hexagonal. En este código, $k = 3 = 6 - 4 + 1 = n - d + 1$ y la cota Singleton es alcanzada. Por lo tanto, $A_4(6,4) = 4^3$.

Definición 2.2.10. (Código MDS) Un código para el cual se alcanza la cota Singleton es llamado separable de máxima distancia, MDS por sus siglas en inglés.

Observación 2.2.2. Ningún código de longitud n y mínima distancia d tiene más palabras código que un código MDS con parámetros n y d; equivalentemente, ningún código de longitud n con M palabras código tiene una distancia mínima mayor que un código MDS con parámetros n y M.

2.3. NP-Completitud del problema de la decodificación de un código lineal binario

Ya sabemos que la decodificación mediante el uso del síndrome se reduce a, dado un vector recibido y, calcular su síndrome y, en la clase lateral correspondiente, encontrar al líder, que va a ser el vector de mínimo peso en la clase. Este problema puede reformularse como el siguiente problema de decisión, para el caso de un código lineal binario.

Problema: Decodificación mediante síndrome.
Entradas: Una matriz binaria H de tamaño $(n-k) \times n$, un vector $s \in \mathbb{F}_2^{n-k}$ y un entero $w > 0$.
Pregunta: ¿Existe un vector $x \in \mathbb{F}_2^n$ de peso $\leq w$, tal que $Hx^T = s$?

Nuestro objetivo es demostrar la NP-Completitud de este problema. Para ello, es necesario que partamos de un problema NP-Completo ya conocido. En nuestro caso, va a ser el del emparejado en 3 dimensiones.

Problema: Emparejado en 3 dimensiones.
Entradas: Un subconjunto $U \subseteq T \times T \times T$, donde T es un conjunto finito.
Pregunta: ¿Existe un subconjunto $W \subseteq U$ tal que $|W| = |T|$, verificando que no hay dos elementos en W que coincidan en alguna coordenada?

Ejemplo 2.3.1. Sea $T = \{1,2,3,4\}$. En primer lugar, consideremos $U = \{(1,2,1),(1,3,2),(2,1,4),(2,2,3),(3,1,1),(4,4,4)\}$. En este caso, la pregunta anterior tiene respuesta afirmativa, pues el subconjunto $W = \{(1,3,2),(2,2,3),(3,1,1),(4,4,4)\}$ cumple las propiedades deseadas. Sin embargo, si ahora consideramos

$U = \{(1,2,1),(1,3,2),(2,1,4),(2,2,3),(3,2,1),(4,4,4)\}$, tenemos que un tal subconjunto no existe.

Observación 2.3.1. Antes de comenzar la demostración, es conveniente codificar el conjunto U de tripletas como una matriz binaria $|U| \times 3|T|$, en la cual cada fila corresponde a una tripleta y tiene peso 3, con cada 1 correspondiendo a una componente de la tripleta en cuestión. Por ejemplo, en el primer caso del ejemplo 2.3.1 tendríamos la siguiente matriz 6×12.

	1	2	3	4	1	2	3	4	1	2	3	4
121:	1	0	0	0	0	1	0	0	1	0	0	0
132:	1	0	0	0	0	0	1	0	0	1	0	0
214:	0	1	0	0	1	0	0	0	0	0	0	1
223:	0	1	0	0	0	1	0	0	0	0	1	0
311:	0	0	1	0	1	0	0	0	1	0	0	0
444	0	0	0	1	0	0	0	1	0	0	0	1

Figura 2.3: Codificación del conjunto U [ERB78]

En términos de esta matriz, una solución al problema del emparejado en 3 dimensiones es la existencia de $|T|$ filas cuyo suma módulo 2 sea $111\cdots1111$.

Ahora ya sí que podemos pasar con la demostración.

Teorema 2.3.1. *El problema de la decodificación mediante síndrome para un código lineal binario es NP-Completo.*

Demostración. En primer lugar, hay que demostrar que es un problema de NP. Para ello, podríamos considerar un algoritmo que primero seleccionara de forma no determinista un peso menor o igual que w y, a continuación, seleccionara también de forma no determinista un vector $x \in \mathbb{F}_2^n$ con tal peso. Pasemos ya a la reducción. Sea una entrada $U \subseteq T \times T \times T$ para el problema del emparejamiento en 3 dimensiones y A la matriz $|U| \times 3|T|$ descrita anteriormente. Podemos entonces ejecutar una instancia del problema de la decodificación con entradas $H = A^T, s = (111\cdots111)$ y $w = |T|$. Si codificamos la solución al problema del emparejamiento W como un vector y de tamaño $|U|$, habiendo un 0 cuando el correspondiente elemento de U no se encuentre en W, y un 1 en caso contrario, tenemos que $x = y$ es una solución factible para el problema de la decodificación. Todas estas transformaciones se pueden hacer en espacio logarítmico, por lo que concluimos que el problema de la decodificación es NP-Completo. □

3 Códigos Reed-Solomon Generalizados

Tras haber realizado un breve recorrido por las bases de la teoría de códigos, pasamos ahora a estudiar una de sus familias más importantes, los códigos cíclicos, muy interesantes desde el punto de vista matemático debido a su correspondencia con los ideales de cocientes de anillos de polinomios. También presentaremos una subfamilia suya, los códigos BCH, que contienen a los códigos Reed-Solomon. A partir de esta última familia construiremos los códigos Reed-Solomon Generalizados, siendo los más relevantes para nuestro estudio. 3.1, 3.2 y 3.3 están basados en [yVP03], mientras que 3.4 en [Hala].

3.1. Códigos Cíclicos

Veamos en primer lugar el concepto de código cíclico.

Definición 3.1.1. Un código lineal C de longitud n sobre $\mathbb{F}_q$ es cíclico si, para cada vector $c = c_0...c_{n-2}c_{n-1}$ en C, el vector $c_{n-1}c_0...c_{n-2}$, obtenido de c mediante el desplazamiento cíclico de coordenadas $i \to i+1$ (mod n), está también en C.

Observación 3.1.1. Tenemos que un código cíclico contiene los n movimientos cíclicos de cualquier palabra código. Por tanto, es conveniente pensar en las posiciones cíclicamente, de manera que cuando se alcance la posición $n-1$, empezamos de nuevo por la posición 0.

Al examinar códigos cíclicos sobre $\mathbb{F}_q$, a menudo representaremos las palabras código en forma de polinomio. Hay una correspondencia biyectiva entre los vectores $c = c_0c_1...c_{n-1}$ en $\mathbb{F}_q^n$ y los polinomios $c(x) = c_0 + c_1x + ... + c_{n-1}x^{n-1} \in \mathbb{F}_q[x]$. En lo que sigue, utilizaremos la notación vectorial c y la polinómica $c(x)$ indistintamente. Se verifica que si $c(x) = c_0 + c_1x + ... + c_{n-1}x^{n-1} \in C$, entonces $xc(x) = c_{n-1}x^n + c_0x + c_1x^2 + ... + c_{n-2}x^{n-1} \in C$, donde hemos desplazado la palabra código cíclicamente una posición a la derecha e hemos igualado x^n a 1. Esto sugiere que el contexto adecuado para estudiar los códigos cíclicos es el anillo cociente $\mathcal{R}_n = \frac{\mathbb{F}_q[x]}{x^n-1}$. Usando la correspondencia de vectores con polinomios presentada, los códigos cíclicos son ideales de $\mathcal{R}_n$ y los ideales de $\mathcal{R}_n$ son códigos cíclicos. Por lo tanto, el estudio de los códigos cíclicos en $\mathbb{F}_q^n$ es equivalente al estudio de los ideales en $\mathcal{R}_n$.

La estructura algebraica de los ideales de $\mathcal{R}_n$ depende de la factorización de $x^n - 1$, por lo que deberemos encontrar sus factores irreducibles. Tenemos dos escenarios posibles: o bien $x^n - 1$ tiene factores irreducibles repetidos o no los tiene. En nuestro estudio, vamos a centrarnos en el último caso. De hecho, por un resultado, sabemos que $x^n - 1$ no tiene factores irreducibles repetidos si y solo si q y n son primos relativos, una hipótesis que mantendremos a partir de ahora. Previo a la prueba del resultado mencionado, necesitamos la siguiente proposición auxiliar.

Proposición 3.1.1. *Sea $f \in \mathbb{F}_q[x]$ y $f' \in \mathbb{F}_q[x]$ la derivada formal de f. Entonces f no tiene factores irreducibles repetidos si y solo si $mcd(f, f') = 1$.*

Proposición 3.1.2. *[Exca] El polinomio $x^n - 1 \in \mathbb{F}_q[x]$ no tiene factores irreducibles repetidos si y solo si $mcd(q, n) = 1$.*

Demostración. Supongamos en primer lugar que $mcd(q, n) \neq 1$. Recordando que q es una potencia de p, tenemos que se da $n = p^e m$, para cierto $e > 0$ y $p \nmid m$. Al estar trabajando sobre un cuerpo de característica p, tenemos que $x^n - 1 = x^{p^e m} - 1 = (x^m - 1)^{p^e}$, que claramente tiene factores irreducibles repetidos. Supongamos ahora que $mcd(q, n) = 1$, y sea $I \subseteq F_q[x]$ el ideal generado por $x^n - 1$ y su derivada formal nx^{n-1}. De $mcd(q, n) = 1$ se sigue que n tiene inversa como elemento de $\mathbb{F}_q$, n^{-1}, luego $n^{-1}nx^{n-1} = x^{n-1} \in I \implies xx^{n-1} = x^n \in I \implies x^n - (x^n - 1) \in I \implies 1 \in I \implies I = (1)$. Finalmente, concluimos que $mcd(x^n - 1, nx^{n-1}) = 1$, por lo que, aplicando la proposición 3.1.1, $x^n - 1$ no tiene factores irreducibles repetidos en $F_q[x]$. □

Para factorizar $x^n - 1$ sobre $\mathbb{F}_q$, es útil encontrar una extensión $\mathbb{F}_{q^t}$ de $\mathbb{F}_q$ que contenga todas sus raíces, entre ellas una n-ésima raíz primitiva de la unidad, que ocurre precisamente cuando $n|(q^t - 1)$, de acuerdo con el Teorema 1.2.12.

Definición 3.1.2. El orden $ord_n(q)$ de q módulo n es el entero positivo más pequeño a tal que $q^a \equiv 1$ (mod n).

Cabe mencionar que si $t = ord_n(q)$, entonces $\mathbb{F}_{q^t}$ contiene una raíz primitiva de la unidad n-ésima α y ninguna extensión más pequeña de $\mathbb{F}_q$ contiene dicha raíz. Por lo tanto, $\mathbb{F}_{q^t}$ es llamado un cuerpo de descomposición de $x^n - 1$ sobre $\mathbb{F}_q$. Tenemos que los factores irreducibles de $x^n - 1$ sobre $\mathbb{F}_q$ deben ser producto de los diferentes polinomios mínimos de las raíces de la unidad n-ésimas en $\mathbb{F}_{q^t}$. Supongamos que γ es un elemento primitivo de $\mathbb{F}_{q^t}$. Entonces $\alpha = \gamma^d$ es una raíz primitiva de la unidad n-ésima, donde $d = \frac{q^t - 1}{n}$. Las raíces de $M_{\alpha^s}(x)$ son $\{\gamma^{ds}, \gamma^{dsq}, \gamma^{dsq^2}, ..., \gamma^{dsq^{r-1}}\} = \{\alpha^s, \alpha^{sq}, \alpha^{sq^2}, ..., \alpha^{sq^{r-1}}\}$, donde r es el entero positivo más pequeño tal que $dsq^r \equiv ds$ (mod $q^t - 1$), por el teorema 1.2.11. Además $dsq^r \equiv ds$ (mod $q^t - 1$) si y solo si $sq^r \equiv s$ (mod n).

Esto nos lleva a extender el concepto de clase lateral q-ciclotómica presentada en la definición 1.2.12.

Definición 3.1.3. Sea s un entero con $0 \leq s < n$. La clase lateral q-ciclotómica de s módulo n es el conjunto $C_s = \{s, sq, ...sq^{r-1}\}$ (mod n), donde r es el entero positivo más pequeño tal que $sq^r \equiv s$ (mod n).

C_s es la órbita de la permutación $i \to iq$ (mod n) que contiene a s. Las distintas clases laterales q-ciclotómicas módulo n forman una partición del conjunto de los enteros $\{0, 1, 2, ..., n - 1\}$. Normalmente denotaremos una clase lateral ciclotómica de esta partición eligiendo s de manera que sea el entero más pequeño contenido en dicha clase. Cabe mencionar que $ord_n(q)$ es el tamaño de la clase lateral q-ciclotómica C_1 módulo n. Podemos ya presentar el siguiente teorema.

Teorema 3.1.3. *Sea n un entero positivo primo relativo con q, $t = ord_n(q)$ y α una raíz primitiva de la unidad n-ésima en $\mathbb{F}_{q^t}$.*

1. *Para todo entero s con $0 \leq s < n$, el polinomio mínimo de α^s sobre $\mathbb{F}_q$ es*

$$M_{\alpha^s}(x) = \prod_{i \in C_s} (x - \alpha^i),$$

donde C_s es la clase lateral q-ciclotómica de s módulo n.

2. *Los conjugados de α^s son los elementos α^i con $i \in C_s$.*

3. *Además*
$$x^n - 1 = \prod_s M_{\alpha^s}(x)$$
es la factorización de $x^n - 1$ en factores irreducibles sobre $\mathbb{F}_q$, donde s está indexado sobre un conjunto de representantes de las clases laterales q-ciclotómicas módulo n.

Ejemplo 3.1.1. Las clases laterales 2-ciclotómicas módulo 9 son $C_0 = \{0\}$, $C_1 = \{1,2,4,8,7,5\}$ y $C_3 = \{3,6\}$. Por tanto, $ord_9(2) = 6$ y las raíces de la unidad primitivas 9-ésimas están en $\mathbb{F}_{64}$ pero no en ninguna extensión más pequeña de $\mathbb{F}_2$. Los factores irreducibles de $x^9 - 1$ sobre $\mathbb{F}_2$ tienen grado 1, 6 y 2. Estos son los polinomios $M_1(x) = x + 1$, $M_\alpha(x)$ y M_{α^3}, donde α es una raíz primitiva de la unidad 9-ésima en $\mathbb{F}_{64}$. El único polinomio irreducible de grado 2 sobre $\mathbb{F}_2$ es $x^2 + x + 1$, por lo que $M_{\alpha^3}(x) = x^2 + x + 1$ (cabe mencionar también que α^3 es una raíz primitiva de la unidad 3-ésima, la cual debe estar en el subcuerpo $\mathbb{F}_4$ de $\mathbb{F}_{64}$). Por lo tanto, la factorización de $x^9 - 1$ es $x^9 - 1 = (x+1)(x^2+x+1)(x^6+x^3+1)$ y $M_\alpha(x) = x^6 + x^3 + 1$.

Como comentamos anteriormente, existe una correspondencia entre los ideales de R_n y los códigos cíclicos de longitud n definidos en el cuerpo en cuestión. Para distinguir el ideal principal $(g(x))$ de $\mathbb{F}_q[x]$ del mismo ideal pero en R_n, usaremos la notación $< g(x) >$ para el ideal principal de R_n generado por $g(x)$. Mostremos ahora que existe una correspondencia biyectiva entre los códigos cíclicos en R_n y los polinomios mónicos divisores de $x^n - 1$.

Teorema 3.1.4. *Sea C un código cíclico diferente de 0 en R_n. Existe un polinomio $g(x) \in C$ con las siguientes propiedades:*

1. *$g(x)$ es el único polinomio mónico de grado mínimo en C.*

2. *$C =< g(x) >$.*

3. *$g(x)|(x^n - 1)$.*
Sea $k = n - gr\ g(x)$ y $g(x) = \sum_{i=0}^{n-k} g_i x^i$, donde $g_{n-k} = 1$. Entonces:

4. *La dimensión de C es k y $\{g(x), xg(x), ..., x^{k-1}g(x)\}$ es una base para C.*

5. *Cada elemento de C es únicamente expresable como un producto $g(x)f(x)$, donde $f(x) = 0$ o $gr\ f(x) < k$.*

6. $$G = \begin{pmatrix} g_0 & g_1 & g_2 & \cdots & g_{n-k} & & 0 \\ 0 & g_0 & g_1 & \cdots & g_{n-k-1} & g_{n-k} & \\ & \cdots & \cdots & \cdots & \cdots & \cdots & \\ 0 & & g_0 & & \cdots & & g_{n-k} \end{pmatrix} \leftrightarrow \begin{pmatrix} g(x) & & & \\ & xg(x) & & \\ & & \cdots & \\ & & & x^{k-1}g(x) \end{pmatrix}$$
es una matriz generadora para C.

7. *Si α es una raíz primitiva de la unidad n-ésima en alguna extensión de $\mathbb{F}_q$, entonces*

$$g(x) = \prod_s M_{\alpha^s}(x),$$

donde el producto es sobre un subconjunto de representantes de las clases laterales q-ciclotómicas módulo n.

Demostración. Sea $g(x)$ un polinomio mónico de grado mínimo en C. Como C no es el código nulo, dicho polinomio existe. Si $c(x) \in C$, entonces por el algoritmo de la división en $\mathbb{F}_q[x]$, $c(x) = g(x)h(x) + r(x)$, donde o bien $r(x) = 0$ o gr r(x) < gr g(x). Como C es un ideal de R_n, $r(x) \in C$ y la minimalidad del grado de $g(x)$ implica que $r(x) = 0$. Esto prueba los puntos 2 y 3. Por el algoritmo de la división, $x^n - 1 = g(x)h(x) + r(x)$, donde de nuevo $r(x) = 0$ o gr r(x) < gr g(x) en $\mathbb{F}_q[x]$. Como $x^n - 1$ corresponde con la palabra cero en C y C es un ideal en R_n, $r(x) \in C$, una contradicción salvo que $r(x) = 0$, demostrando el punto 3. Supongamos que gr $g(x) = n - k$. Por los puntos 2 y 3, si $c(x) \in C$ con $c(x) = 0$ o gr $c(x) < n$, entonces $c(x) = g(x)f(x)$ en $\mathbb{F}_q[x]$. Si $c(x) = 0$, entonces $f(x) = 0$. Si $c(x) \neq 0$, gr $c(x) < n$ implica que gr $f(x) < k$, pues el grado del producto de dos polinomios es la suma de ambos. Por lo tanto

$$C = \{g(x)f(x) | f(x) = 0 \text{ o gr f(x)} < \text{k}\}.$$

En consecuencia, C tiene dimensión como mucho k y

$$\{g(x), xg(x), ..., x^{k-1}g(x)\}$$

genera C. Ya que estos k polinomios son de diferentes grados, también son independientes en $\mathbb{F}_q[x]$. Al ser de grado como mucho $n - 1$ permanecen independientes en R_n, dándonos los puntos 4 y 5. Las palabras código en esta base, escritas como n-tuplas, dan G en el punto 6. El punto 7 se obtiene del teorema 3.1.3. □

Observación 3.1.2. Cabe mencionar que el punto 2 muestra que R_n es un dominio de ideales principales. Los puntos 1-6 del teorema 3.1.4 siguen siendo ciertos incluso con $mcd(n, q) \neq 1$. Sin embargo, el punto 7 sí requiere de $mcd(n, q) = 1$.

Corolario 3.1.1. *Sea C un código cíclico diferente de 0 en R_n. Las siguientes afirmaciones son equivalentes:*

1. *$g(x)$ es el polinomio mónico de mínimo grado en C.*
2. *$C =< g(x) >$, $g(x)$ es mónico y $g(x)|(x^n - 1)$.*

Demostración. Que 1 implique 2 fue demostrado en la prueba del teorema 3.1.4. Supongamos por tanto que se verifica 2. Sea $g_1(x)$ el polinomio mónico de mínimo grado en C. De la prueba de los puntos 1 y 2 del teorema 3.1.4, $g_1(x)|g(x)$ en $\mathbb{F}_q[x]$ y $C =< g_1(x) >$. Como $g_1(x) \in C =< g(x) >$, $g_1(x) \equiv g(x)a(x) \pmod{x^n - 1}$ implica $g_1(x) = g(x)a(x) + (x^n - 1)b(x)$ en $\mathbb{F}_q[x]$. Al darse $g(x)|(x^n - 1)$, $g(x)|(g(x)a(x) + (x^n - 1)b(x) = g_1(x))$. Tanto $g_1(x)$ como $g(x)$ son mónicos y se dividen mutuamente en $\mathbb{F}_q[x]$, luego tenemos que son iguales. □

El siguiente corolario del teorema 3.1.4 muestra la relación entre los polinomios generadores de dos códigos cíclicos cuando uno es un subcódigo del otro.

Corolario 3.1.2. *Sean C_1 y C_2 códigos cíclicos sobre $\mathbb{F}_q$ con polinomios generadores $g_1(x)$ y $g_2(x)$, respectivamente. Entonces $C_1 \subseteq C_2$ si y solo si $g_2(x)|g_1(x)$.*

El teorema 3.1.4 muestra que existe un polinomio mónico $g(x)$ dividiendo a $x^n - 1$ y generando C. El corolario 3.1.2 muestra que el polinomio mónico dividiendo a $x^n - 1$ que genera C es único, siendo conocido como el polinomio generador del código cíclico C.

Más allá del polinomio generador, hay muchos polinomios que pueden ser usados para generar un código cíclico. Un elemento e de un anillo satisfaciendo $e^2 = e$ es llamado un idempotente. En el siguiente teorema probamos que cada código cíclico en R_n contiene un único idempotente que lo genera, conocido como el idempotente generador del código.

Teorema 3.1.5. *Sea C un código cíclico en R_n. Entonces:*

1. *Existe un único idempotente $e(x) \in C$ tal que $C =< e(x) >$.*
2. *Si $e(x)$ es un idempotente diferente de cero en C, entonces $C =< e(x) >$ si y solo si $e(x)$ es un 1 de C.*

Demostración. Si C es el código cero, entonces el idempotente es el polinomio cero, 1) es claro y 2) no aplica.
Asumimos por tanto que C no es cero. Demostramos 2) en primer lugar. Supongamos que $e(x)$ es un 1 en C. Entonces $< e(x) > \subseteq C$ ya que C es un ideal. Si $c(x) \in C$, entonces $c(x)e(x) = c(x)$ en C, lo que implica que $< e(x) >= C$. Recíprocamente, supongamos que $e(x)$ es un idempotente diferente de cero tal que $C =< e(x) >$. Entonces cada elemento $c(x) \in C$ puede ser escrito de la forma $c(x) = f(x)e(x)$, pero $c(x)e(x) = f(x)(e(x))^2 = f(x)e(x) = c(x)$, lo que implica que $e(x)$ es un 1 de C.
Como C es diferente de cero, por 2) si $e_1(x)$ y $e_2(x)$ son generadores idempotentes entonces ambos son 1s y $e_1(x) = e_2(x)e_1(x) = e_2(x)$, por lo que solo necesitamos demostrar que un idempotente generador existe. Si $g(x)$ es el polinomio generador para C entonces $g(x)|(x^n - 1)$ por el Teorema 3.1.4. Sea $h(x) = \frac{x^n-1}{g(x)}$. Entonces $mcd(g(x), h(x)) = 1$ en $\mathbb{F}_q[x]$ ya que $x^n - 1$ tiene raíces distintas. Por el algoritmo de Euclides existen polinomios $a(x), b(x) \in \mathbb{F}_q[x]$ de manera que $a(x)g(x) + b(x)h(x) = 1$. Sea $e(x) \equiv a(x)g(x) \bmod x^n - 1)$, es decir, $e(x)$ es el representante de la clase lateral de $a(x)g(x) + (x^n - 1)$ en R_n. Entonces en R_n, $e(x)^2 \equiv (a(x)g(x))(1 - b(x)h(x)) \equiv a(x)g(x) \equiv e(x) \pmod{x^n - 1}$, ya que $g(x)h(x) = x^n - 1$. También si $c(x) \in C, c(x) = f(x)g(x)$, implicando que $c(x)e(x) \equiv f(x)g(x)(1 - b(x)h(x)) \equiv f(x)g(x) \equiv c(x) \pmod{x^n - 1}$, por lo que $e(x)$ es un 1 en C, y 1) sigue de 2). □

Si C_1 y C_2 son códigos de longitud n sobre $\mathbb{F}_q$, entonces $C_1 + C_2 = \{c_1 + c_2 | c_1 \in C_1, c_2 \in C_2\}$ es la suma de C_1 y C_2. Tanto la intersección como la suma de dos códigos cíclicos son cíclicas, y sus polinomios e idempotentes generadores son determinados en el siguiente teorema.

Teorema 3.1.6. *Sea C_i un código cíclico de longitud n sobre $\mathbb{F}_q$ con polinomio generador $g_i(x)$ e idempotente generador $e_i(x)$, para $i = 1$ e $i = 2$. Entonces $C_1 + C_2$ tiene polinomio generador $mcd(g_1(x), g_2(x))$ e idempotente generador $e_1(x) + e_2(x) - e_1(x)e_2(x)$.*

Demostración. Sea $g(x) = mcd(g_1(x), g_2(x))$. Se sigue del algoritmo de Euclides que $g(x) = g_1(x)a(x) + g_2(x)b(x)$ para ciertos $a(x)$ y $b(x)$ en $\mathbb{F}_q[x]$. Por lo tanto, $g(x) \in C_1 + C_2$. Ya que $C_1 + C_2$ es cíclico, $< g(x) > \subseteq C_1 + C_2$. Por otro lado, $g(x)|g_1(x)$, lo que demuestra que $C_1 \subseteq < g(x) >$ por el corolario 3.1.2; similarmente, $C_2 \subseteq < g(x) >$ implicando que $C_1 + C_2 \subseteq < g(x) >$. En consecuencia, $C_1 + C_2 = < g(x) >$. Puesto que $g(x)|(x^n - 1)$ ya que $g(x)|g_1(x)$ y $g(x)$ es mónico, $g(x)$ es el polinomio generador de $C_1 + C_2$ por el corolario 3.1.1. Si $c(x) = c_1(x) + c_2(x)$ donde $c_i(x) \in C_i$ para $i = 1$ e $i = 2$, entonces
$c(x)(e_1(x) + e_2(x) - e_1(x)e_2(x)) =$
$c_1(x) + c_1(x)e_2(x) - c_1(x)e_2(x) + c_2(x)e_1(x) + c_2(x) - c_2(x)e_1(x) = c(x)$. El resultado se sigue del teorema 3.1.5 porque $e_1(x) + e_2(x) - e_1(x)e_2(x) \in C_1 + C_2$. □

Podemos ya presentar un subconjunto de los idempotentes llamados idempotentes primitivos que, una vez conocidos, producirán todos los idempotentes en R_n y, en consecuencia, todos los códigos cíclicos. Sea $x^n - 1 = f_1(x) \cdots f_s(x)$, donde $f_i(x)$ es irreducible sobre $\mathbb{F}_q$ para $1 \leq i \leq s$. Los $f_i(x)$ son distintos ya que $x^n - 1$ tiene raíces distintas. Sea $\hat{f}_i = \frac{x^n-1}{f_i(x)}$. En el siguiente teorema demostraremos que los ideales $< \hat{f}_i(x) >$ de R_n son los ideales minimales de R_n. Recordemos que un ideal I en un anillo R es un ideal minimal si no hay ideales propios entre $\{0\}$ e I. Denotamos al idempotente generador de $< \hat{f}_i(x) >$ por $\hat{e}_i(x)$. Los idempotentes $\hat{e_1}(x), ..., \hat{e_s}(x)$ son llamados los idempotentes primitivos de R_n.

Teorema 3.1.7. *Lo siguiente se verifica en R_n.*

1. *Los ideales $< \hat{f}_i(x) >$ para $1 \leq i \leq s$ son todos los ideales minimales de R_n.*
2. *R_n es el espacio vectorial suma directa de $< \hat{f}_i(x) >$ para $1 \leq i \leq s$.*
3. *Si $i \neq j$ entonces $\hat{e}_i(x)\hat{e}_j(x) = 0$ en R_n.*
4. *$\sum_{i=1}^{s} \hat{e}_i(x) = 1$ en R_n.*
5. *Los únicos idempotentes en $< \hat{f}_i(x) >$ son 0 y $\hat{e}_i(x)$.*
6. *Si $e(x)$ es un idempotente distinto de cero en R_n, entonces hay un subconjunto T de $\{1, 2, ..., s\}$ tal que $e(x) = \sum_{i \in T} \hat{e}_i(x)$ y $< e(x) >= \sum_{i \in T} < \hat{f}_i(x) >$.*

Demostración. Supongamos que $< \hat{f}_i(x) >$ no es un ideal minimal de R_n. Por el corolario 3.1.2 existiría un polinomio generador $g(x)$ de un ideal distinto de cero, propiamente contenido en $< \hat{f}_i(x) >$, tal que $\hat{f}_i(x)|g(x)$, con $g(x) \neq \hat{f}_i(x)$. Como $f_i(x)$ es irreducible y $g(x)|(x^n - 1)$, esto es imposible. Por lo tanto, $< \hat{f}_i(x) >$ es un ideal minimal de R_n, completando parte de 1).

Como $\{\hat{f}_i(x) | 1 \leq i \leq s\}$ no tiene ningún factor irreducible común con $x^n - 1$ y cada polinomio del conjunto divide a $x^n - 1$, $mcd(\hat{f}_1(x), ..., \hat{f}_s(x)\} = 1$. Aplicando el algoritmo de Euclides inductivamente

$$1 = \sum_{i=1}^{s} a_i(x)\hat{f}_i(x) \tag{3.1}$$

para ciertos $a_i(x) \in \mathbb{F}_q[x]$. Por lo tanto el 1 está en la suma de los ideales $< \hat{f}_i(x) >$, que es a su vez un ideal de R_n. En cualquier anillo, el único ideal que contiene al 1 del anillo es el propio anillo. Esto demuestra que R_n es el espacio vectorial suma de los ideales $< \hat{f}_i(x) >$.

Para demostrar que es una suma directa, debemos probar que
$< \hat{f}_i(x) > \cap \sum_{j \neq i} < \hat{f}_j(x) >= \{0\}$ para $1 \leq i \leq s$. Como $f_i(x) | \hat{f}_j(x)$ para $j \neq i$, $f_j(x) \nmid \hat{f}_j(x)$, y los factores irreducibles de $x^n - 1$ son diferentes, concluimos que
$f_i(x) = mcd\{\hat{f}_j(x) | 1 \leq j \leq s, j \neq i\}$. Aplicando inducción a los resultados del teorema 3.1.6 se demuestra que $< f_i(x) >= \sum_{j \neq i} < \hat{f}_j(x) >$. Por lo tanto, $< \hat{f}_i(x) > \cap \sum_{j \neq i} < \hat{f}_j(x) >=<$ $\hat{f}_i(x) > \cap < f_i(x) >=< mcm(\hat{f}_i(x), f_i(x)) >=< x^n - 1 >= \{0\}$ por el teorema 3.1.6, completando 2). Sea $\mathcal{M} =< m(x) >$ un ideal minimal cualquiera de R_n. Ya que

$$0 \neq m(x) = m(x) \cdot 1 = \sum_{i=1}^{s} m(x) a_i(x) \hat{f}_i(x),$$

por (3.1) existe un i tal que $m(x)a_i(x)\hat{f}_i(x) \neq 0$. Por tanto $\mathcal{M} \cap < \hat{f}_i(x) > \neq 0$, puesto que $m(x)a_i(x)\hat{f}_i(x) \in \mathcal{M} \cap < \hat{f}_i(x) >$ y en consecuencia $\mathcal{M} =< \hat{f}_i(x) >$ por minimalidad de $\mathcal{M}$ y $< \hat{f}_i(x) >$. Esto completa la prueba de 1). Si
$i \neq j, \hat{e}_i(x)\hat{e}_j(x) \in < \hat{f}_i(x) > \cap < \hat{f}_j(x) >= \{0\}$ por 2), dándonos 3). Usando 3) y aplicando inducción al teorema 3.1.6, $\sum_{i=1}^{s} \hat{e}_i(x)$ es el idempotente generador de $\sum_{i=1}^{s} < \hat{f}_i(x) > = R_n$ por el punto 2). El polinomio generador de R_n es 1, verificando 4). Si $e(x)$ es un idempotente diferente de cero en $< \hat{f}_i(x) >$ entonces $< e(x) >$ es un ideal contenido en $< \hat{f}_i(x) >$. Por minimalidad, ya que $e(x)$ es diferente de cero, $< \hat{f}_i(x) >=< e(x) >$, implicando por el teorema 3.1.5 que $e(x) = \hat{e}_i(x)$, ya que ambos son el único 1 de $< \hat{f}_i(x) >$. Se verifica por tanto 5).
Para 6) cabe notar que $e(x)\hat{e}_i(x)$ es un idempotente en $< \hat{f}_i(x) >$. Por tanto, o bien $e(x)\hat{e}_i(x)$ es 0 o $\hat{e}_i(x)$ por 5). Sea $T = \{i | e(x)\hat{e}_i(x) \neq 0\}$. Entonces por el punto 4),
$e(x) = e(x) \cdot 1 = e(x) \sum_{i=1}^{s} \hat{e}_i(x) = \sum_{i=1}^{s} e(x)\hat{e}_i(x) = \sum_{i \in T} \hat{e}_i(x)$. Además,
$< e(x) >=< \sum_{i \in T} \hat{e}_i(x) >= \sum_{i \in T} < \hat{e}_i(x) >$ por el teorema 3.1.6 e inducción. □

Veamos ahora una permutación concreta que lleva idempotentes de R_n a idempotentes de R_n. Sea a un entero tal que $mcd(a, n) = 1$. La función μ_a definida en $\{0, 1, ..., n-1\}$ por $i\mu_a \equiv ia \pmod{n}$ es una permutación de las posiciones $\{0, 1, ..., n-1\}$ de un código cíclico de longitud n y es llamado un multiplicador. Ya que los códigos cíclicos de longitud n son representados como ideales en R_n, para $a > 0$ es conveniente mirar μ_a actuando en R_n como

$$f(x)\mu_a \equiv f(x^a) (\text{mod } x^n - 1). \tag{3.2}$$

Esta ecuación es consistente con la definición original de μ_a, porque $x^i\mu_a = x^{ia} = x^{ia+jn}$ en R_n, para un entero j tal que $0 \leq ia + jn < n$ ya que $x^n = 1$ en R_n. Es decir, $x^i\mu_a = x^{ia \bmod n}$. Si $a < 0$ podemos darle significado a $f(x^a)$ en R_n definiendo $x^i\mu_a = x^{ia \bmod n}$ donde, por supuesto, $0 \leq ia \bmod n < n$. Con esta interpretación, (3.2) es consistente con la definición original de μ_a cuando $a < 0$.

Por el teorema 3.1.3, si $t = ord_n(q)$, entonces $\mathbb{F}_{q^t}$ es un cuerpo de descomposición de $x^n - 1$; por tanto, $\mathbb{F}_{q^t}$ contiene una raíz primitiva de la unidad n-ésima α, y $x^n - 1 = \prod_{i=0}^{n-1}(x - \alpha^i)$ es la factorización de $x^n - 1$ en factores lineales sobre $\mathbb{F}_{q^t}$. Además, $x^n - 1 = \prod_s M_{\alpha^s}(x)$ es la factorización de $x^n - 1$ en factores irreducibles sobre $\mathbb{F}_q$, donde s está indexado sobre un conjunto de representantes de las clases laterales q-ciclotómicas módulo n.

Sea C un código cíclico en R_n con polinomio generador $g(x)$. Por los teoremas 3.1.3 1) y 3.1.4 7), $g(x) = \prod_s M_{\alpha^s}(x) = \prod_s \prod_{i \in C_s}(x - \alpha^i)$, donde s está indexado sobre algún

subconjunto de representantes de las clases laterales q-ciclotómicas módulo n C_s.

Definición 3.1.4. (Conjunto de Definición) El conjunto de definición de C se define como $T = \cup_s C_s$, es decir, la unión de las clases laterales q-ciclotómicas anteriores.

Observación 3.1.3. Si cambiamos la raíz primitiva de la unidad n-ésima, cambiamos también T, por lo que T es calculado en función de una raíz primitiva fijada.

Definición 3.1.5. (Ceros) Las raíces de la unidad $\mathcal{Z} = \{\alpha^i | i \in T\}$ son llamadas los ceros de C, y $\{\alpha^i | i \notin T\}$ los no ceros.

Se deduce que $c(x)$ pertenecerá a C si y solo si $c(\alpha^i) = 0$ para cada $i \in T$, por el teorema 3.1.4. Cabe mencionar que T determina completamente el polinomio generador $g(x)$. Por el teorema 3.1.4, la dimensión de C es $n - |T|$, ya que $|T|$ es el grado de $g(x)$.

Ejemplo 3.1.2. A continuación presentamos una tabla dando la dimensión, polinomios generadores $g_i(x)$ y conjunto de definición (relativo a la raíz primitiva α que verifica $\alpha^3 = \alpha + 1$) de todos los códigos cíclicos C_i de longitud 7 sobre $\mathbb{F}_2$.

i	dim	$g_i(x)$	Conjunto de definición
0	0	$1 + x^7$	$\{0, 1, 2, 3, 4, 5, 6\}$
1	1	$1 + x + x^2 + ... + x^6$	$\{1, 2, 3, 4, 5, 6\}$
2	3	$1 + x^2 + x^3 + x^4$	$\{0, 1, 2, 4\}$
3	3	$1 + x + x^2 + x^4$	$\{0, 3, 5, 6\}$
4	4	$1 + x + x^3$	$\{1, 2, 4\}$
5	4	$1 + x^2 + x^3$	$\{3, 5, 6\}$
6	6	$1 + x$	$\{0\}$
7	7	1	$\emptyset$

El siguiente teorema da algunas propiedades básicas de los códigos cíclicos en relación con sus conjuntos de definición.

Teorema 3.1.8. *Sea α una raíz primitiva de la unidad n-ésima en alguna extensión de $\mathbb{F}_q$. Sea C un código cíclico de longitud n sobre $\mathbb{F}_q$ con conjunto de definición T y polinomio generador $g(x)$. Se cumple lo siguiente:*

1. *T es una unión de clases laterales q-ciclotómicas módulo n.*
2. *$g(x) = \prod_{i \in T}(x - \alpha^i)$.*
3. *$c(x) \in R_n$ está en C si y solo si $c(\alpha^i) = 0$ para todo $i \in T$.*
4. *La dimensión de C es $n - |T|$.*

Definición 3.1.6 (Complementario de un código)**.** Sea C un código de longitud n sobre $\mathbb{F}_q$. Definimos un complementario de C como un código C^C tal que $C + C^C = \mathbb{F}_q^n$ y $C \cap C^C = \{0\}$.

Observación 3.1.4. En general, el complementario no es único, pero sí en el caso cíclico, donde además tenemos garantizado que se preserva la propiedad de ser cíclico. En el siguiente teorema damos el polinomio generador de dicho complementario.

Teorema 3.1.9. *Sea C un código cíclico de longitud n sobre $\mathbb{F}_q$ con polinomio generador $g(x)$, idempotente generador $e(x)$ y conjunto de definición T. Sea C^C el complementario cíclico de C. Se tiene lo siguiente:*

1. *$h(x) = \frac{x^n-1}{g(x)}$ es el polinomio generador de C^C y $1 - e(x)$ es su idempotente generador.*
2. *C^C es la suma de los ideales minimales de R_n no contenidos en C.*
3. *Si $\mathcal{N} = \{0, 1, ..., n-1\}$, entonces $\mathcal{N} \setminus T$ es el conjunto de definición de C^C.*

Presentamos ahora el polinomio e idempotente generadores del dual de un código cíclico.

Teorema 3.1.10. *Sea C un $[n, k]$ código cíclico sobre $\mathbb{F}_q$ con polinomio generador $g(x)$, idempotente generador $e(x)$ y conjunto de definición T. Sea $h(x) = \frac{x^n-1}{g(x)}$. Lo siguiente se verifica:*

1. *$C^\perp$ es un código cíclico y $C^\perp = C^C \mu_{-1}$.*
2. *$C^\perp$ tiene idempotente generador $1 - e(x)\mu_{-1}$ y polinomio generador*

$$\frac{x^k}{h(0)} h(x^{-1}).$$

3. *Si $\beta_1, ..., \beta_k$ son los ceros de C entonces $\beta_1^{-1}, ..., \beta_k^{-1}$ son los no ceros de $C^\perp$.*
4. *Si $\mathcal{N} = \{0, 1, ..., n-1\}$ entonces $\mathcal{N} \setminus (-1)T$ mod n es el conjunto de definición de $C^\perp$.*
5. *Precisamente uno de entre C y $C^\perp$ es de tipo impar y el otro es de tipo par.*

En cualquier código es importante ser capaces de determinar la distancia mínima, clave para conocer su capacidad correctora de errores, siendo útil por tanto disponer de cotas sobre dicha distancia, especialmente cotas inferiores. Una de las más conocidas es la de Bose-Ray-Chaudhuri-Hocquenghem, abreviada como cota BCH. Esta cota depende de los ceros del código y especialmente en la habilidad para encontrar cadenas de ceros "consecutivos".

Antes de proceder con la cota BCH vamos a presentar un lema usado en su demostración y útil en más contextos, sobre el determinante de una matriz de Vandermonde.

Definición 3.1.7. (Matriz de Vandermonde) Sean $\alpha_1, ..., \alpha_s$ elementos de un cuerpo $\mathbb{F}$. La matriz $s \times s$ $V = [v_{ij}]$, donde $v_{i,j} = \alpha_j^{i-1}$, es llamada una matriz de Vandermonde.

Lema 3.1.11. *$detV = \prod_{1 \le i < j \le s} (\alpha_j - \alpha_i)$. En particular, V no es singular si los elementos $\alpha_1, ..., \alpha_s$ son diferentes.*

Sea C un código cíclico de longitud n sobre $\mathbb{F}_q$ y α una raíz primitiva de la unidad n-ésima en $\mathbb{F}_{q^t}$, donde $t = ord_n(q)$. Recordemos que T es un conjunto de definición para C supuesto que los ceros de C son $\{\alpha^i | i \in T\}$. Por lo tanto, T debe ser unión de clases laterales q-ciclotómicas módulo n. Decimos que T contiene un conjunto de s elementos consecutivos $\mathcal{S}$ supuesto que existe un conjunto $\{b, b+1, ..., b+s-1\}$ de s enteros consecutivos tal que $\{b, b+1, ..., b+s-1\}$ mod n $= \mathcal{S} \subseteq T$.

Ejemplo 3.1.3. Consideramos el código binario cíclico C de longitud 7 con conjunto de definición $T = \{0, 3, 6, 5\}$. Entonces, T tiene un conjunto de tres elementos consecutivos $\mathcal{S} = \{5, 6, 0\}$.

Teorema 3.1.12. *(Cota BCH) Sea C un código cíclico de longitud n sobre $\mathbb{F}_q$ con conjunto de definición T. Supongamos que C tiene peso mínimo d y que T contiene $\delta - 1$ elementos consecutivos, para algún entero δ. Entonces $d \geq \delta$.*

Demostración. Por hipótesis, C tiene ceros que incluyen $\alpha^b, \alpha^{b+1}, ..., \alpha^{b+\delta-2}$. Sea $c(x)$ una palabra código distinta de 0 en C de peso w, y sea

$$c(x) = \sum_{j=1}^{w} c_{i_j} x^{i_j}.$$

Supongamos por el contrario que $w < \delta$. Como $c(\alpha^i) = 0$ para $b \leq i \leq b + \delta - 2$, $Mu^T = 0$, donde

$$M = \begin{pmatrix} \alpha^{i_1 b} & \alpha^{i_2 b} & \dots & \alpha^{i_w b} \\ \alpha^{i_1 (b+1)} & \alpha^{i_2 (b+1)} & \dots & \alpha^{i_w (b+1)} \\ & & \vdots & \\ \alpha^{i_1 (b+w-1)} & \alpha^{i_2 (b+w-1)} & \dots & \alpha^{i_w (b+w-1)} \end{pmatrix}$$

y $u = c_{i_1} c_{i_2} ... c_{i_w}$. Ya que $u \neq 0$, M es una matriz singular y por tanto $\det M = 0$. Pero $\det M = \alpha^{(i_1 + i_2 + ... + i_w) b} \cdot \det V$, donde V es la matriz de Vandermonde

$$V = \begin{pmatrix} 1 & 1 & \dots & 1 \\ \alpha^{i_1} & \alpha^{i_2} & \dots & \alpha^{i_w} \\ & & \vdots & \\ \alpha^{i_1 (w-1)} & \alpha^{i_2 (w-1)} & \dots & \alpha^{i_w (w-1)} \end{pmatrix}.$$

Ya que los α^{i_j} son distintos, $\det V \neq 0$ por el lema 3.1.11, contradiciendo $\det M = 0$. □

Ejemplo 3.1.4. Sea C el código binario cíclico $[31, 25, d]$ con conjunto de definición $T = \{0, 3, 6, 12, 24, 17\}$. Si aplicamos la cota BCH a C tenemos que $d \geq 2$ ya que el conjunto consecutivo más largo en T es de tamaño 1.

3.2. Códigos BCH

Los códigos BCH son códigos cíclicos diseñados para sacar provecho de la cota BCH. Nos gustaría construir un código cíclico C de longitud n sobre $\mathbb{F}_q$ que tenga simultáneamente un peso mínimo alto y dimensión alta. Tener un peso mínimo alto, gracias a la cota BCH, puede ser solventado escogiendo un conjunto de definición T con un número grande de elementos consecutivos. Ya que la dimensión de C es $n - |T|$ por el teorema 3.1.8, nos gustaría que $|T|$ fuera lo más pequeño posible. Por lo tanto, si quisiéramos que C tuviera distancia mínima al menos δ, podemos escoger un conjunto de definición tan pequeño como sea posible que sea unión de clases laterales q-ciclotómicas con $\delta - 1$ elementos consecutivos.

Definición 3.2.1. (Código BCH) Sea δ un entero con $2 \leq \delta \leq n$. Un código BCH C sobre $\mathbb{F}_q$ de longitud n y distancia diseñada δ es un código cíclico con conjunto de definición

$$T = C_b \cup C_{b+1} \cup ... \cup C_{b+\delta-2}$$

donde C_i es la clase lateral q-ciclotómica módulo n conteniendo i.

Por la cota BCH, este código tiene distancia mínima de al menos δ.

Teorema 3.2.1. *Un código BCH de distancia diseñada δ tiene un peso mínimo de al menos δ.*

Demostración. El conjunto de definición de la definición 3.2.1 contiene $\delta - 1$ elementos consecutivos. El resultado se sigue de la cota BCH. □

Veamos un ejemplo de construcción de un código BCH.

Ejemplo 3.2.1. Supongamos que queremos construir un código BCH sobre $\mathbb{F}_2$ de longitud 9 y distancia diseñada 3. Por el ejemplo 3.1.1, conocemos las clases laterales 2-ciclotómicas módulo 9, y si $b = 0$, tenemos que $b + 3 - 2 = 1$, luego el conjunto de definición del código será $T = C_0 \cup C_1 = \{0, 1, 2, 4, 5, 7, 8\}$.

3.3. Códigos Reed-Solomon

Pasamos ahora a definir los códigos Reed-Solomon como una subfamilia de los códigos BCH.

Definición 3.3.1. (Código Reed-Solomon) Un código Reed-Solomon (abreviado como código RS) C sobre $\mathbb{F}_q$ es un código BCH de longitud $n = q - 1$.

Por lo tanto, tenemos que $ord_n(q) = 1$, implicando que todos los factores irreducibles de $x^n - 1$ son de grado 1 y todas las clases laterales q-ciclotómicas módulo n tienen tamaño 1. De hecho, las raíces de $x^n - 1$ son exactamente los elementos diferentes de cero en $\mathbb{F}_q$ y una raíz primitiva de la unidad n-ésima es un elemento primitivo de $\mathbb{F}_q$. En consecuencia, si C tiene distancia diseñada δ, el conjunto de definición de C tiene tamaño $\delta - 1$ y es $T = \{b, b + 1, ..., b + \delta - 2\}$, para algún entero b. Por el teorema 3.2.1 y la cota Singleton, la dimensión k y distancia mínima d de C satisfacen $k = n - \delta + 1 \geq n - d + 1 \geq k$, alcanzándose la igualdad en ambas. En particular, C es MDS. El siguiente teorema recoge estas propiedades:

Teorema 3.3.1. *Sea C un código RS sobre $\mathbb{F}_q$ de longitud $n = q - 1$ y distancia diseñada δ. Entonces:*

1. *C tiene como conjunto de definición $T = \{b, b + 1, ..., b + \delta - 2\}$ para algún entero b.*
2. *C tiene distancia mínima $d = \delta$ y dimensión $k = n - d + 1$.*
3. *C es MDS.*

Veamos que el dual y el complementario de un código RS siguen siendo RS. Supongamos que T es el conjunto de definición de un código RS C de longitud n y distancia diseñada δ. Entonces, T es un conjunto con $\delta - 1$ elementos consecutivos de $\mathcal{N} = \{0, 1, ..., n - 1\}$. Por el teorema 3.1.9, el conjunto de definición del complementario cíclico C^C de C es $\mathcal{N} \setminus T$, el cual es un conjunto de $n - \delta + 1$ elementos consecutivos, implicando que C^C es RS. De manera

análoga, como $(-1)T$ mod n es también un conjunto de $\delta - 1$ elementos consecutivos de $\mathcal{N}$ tenemos que, usando el teorema 3.1.10, el conjunto de definición $\mathcal{N} \setminus (-1)T$ mod n de $C^{\perp}$ es un conjunto consecutivo de $n - \delta + 1$ elementos. Por tanto, $C^{\perp}$ es un código RS.

Ejemplo 3.3.1. Un elemento primitivo de $\mathbb{F}_{13}$ es 2. Sea C el código RS de distancia diseñada 5 sobre $\mathbb{F}_{13}$. Es un código de longitud 12 con conjunto de definición $\{1,2,3,4\}$ y polinomio generador $(x-2)(x-2^2)(x-2^3)(x-2^4) = 10 + 2x + 7x^2 + 9x^3 + x^4$. Por el teorema 3.3.1, C tiene distancia mínima 5 y C es un código MDS $[12,8,5]$. $C^{\perp}$ es el código RS [12,4,9] con conjunto de definición $\{0,1,2,3,4,5,6,7\}$ y polinomio generador
$(x-2^0)(x-2^1)(x-2^2)\cdots(x-2^7) = 3 + 12x + x^2 + 5x^3 + 11x^4 + 4x^5 + 10x^6 + 5x^7 + x^8$. El complementario cíclico de C es el código RS $[12,4,9]$ con conjunto de definición $\{5,6,7,8,9,10,11,0\}$ y polinomio generador
$(x-2^0)(x-2^5)(x-2^6)\cdots(x-2^{11}) = 9 + 6x + 12x^2 + 10x^3 + 8x^4 + 6x^5 + 9x^6 + 4x^7 + x^8$.

Presentamos un teorema que nos va a dar una formulación diferente de los códigos RS, que nos será muy útil más tarde para definir los dos tipos de códigos que se usan en el trabajo para el criptosistema de McEliece. Para $k \geq 0$, sea $\mathcal{P}_k$ el conjunto de polinomios de grado menor que k, incluyendo el polinomio cero, en $\mathbb{F}_q[x]$. $\mathcal{P}_0$ denotará al polinomio cero.

Teorema 3.3.2. *Sea α un elemento primitivo de $\mathbb{F}_q$ y k un entero con $0 \leq k \leq n = q-1$. Entonces*

$$C = \{(f(1), f(\alpha), f(\alpha^2), ..., f(\alpha^{q-2})) | f \in \mathcal{P}_k\}$$

es el código RS $[n, k, n-k+1]$ sobre $\mathbb{F}_q$.

Demostración. Claramente C es un código lineal sobre $\mathbb{F}_q$, ya que $\mathcal{P}_k$ es un subespacio lineal sobre $\mathbb{F}_q$ de $\mathbb{F}_q[x]$. Ya que $\mathcal{P}_k$ es k-dimensional, C será también k-dimensional si podemos demostrar que si f y f_1 son elementos diferentes de $\mathcal{P}_k$, entonces los correspondientes elementos en C son diferentes. Si los últimos son iguales, entonces su diferencia es $\{0\}$, implicando que $f - f_1$, el cual es un polinomio no nulo de grado como mucho $k-1$, tiene $n \geq k$ raíces, entrando en contradicción con el hecho de que un polinomio sobre $\mathbb{F}_q$ de grado n tiene como mucho n raíces en cualquier extensión de $\mathbb{F}_q$. En consecuencia, C es k-dimensional.
Sea D el código RS $[n, k, n-k+1]$ sobre $\mathbb{F}_q$. Por lo tanto, D tiene conjunto de definición $T = \{1, 2, ..., n-k\}$. Probemos que $C = D$; basta con demostrar que $C \subseteq D$ ya que ambos códigos son k-dimensionales. Sea $c(x) = \sum_{j=0}^{n-1} c_j x^j \in C$. Entonces existe algún $f(x) = \sum_{m=0}^{k-1} f_m x^m \in \mathcal{P}_k$ tal que $c_j = f(\alpha^j)$ para $0 \leq j < n$. Para demostrar que $c(x) \in D$ necesitamos probar que $c(\alpha^i) = 0$ para $i \in T$ por el teorema 3.1.8. Si $i \in T$, entonces

$$c(\alpha^i) = \sum_{j=0}^{n-1} c_j \alpha^{ij} = \sum_{j=0}^{n-1} (\sum_{m=0}^{k-1} f_m \alpha^{jm}) \alpha^{ij} = \sum_{m=0}^{k-1} f_m \sum_{j=0}^{n-1} \alpha^{(i+m)j} = \sum_{m=0}^{k-1} f_m \frac{\alpha^{(i+m)n} - 1}{\alpha^{i+m} - 1}.$$

Pero $\alpha^{(i+m)n} = 1$ y $\alpha^{i+m} \neq 1$ ya que $1 \leq i + m \leq n - 1 = q - 2$ y α es una raíz primitiva de la unidad n-ésima. Por lo tanto, $c(\alpha^i) = 0$ para $i \in T$, lo que implica que $C \subseteq D$. En consecuencia, $C = D$. □

Cabe mencionar que el código RS definido de esta manera alternativa con $k = 0$ es precisamente el código cero.

Esta formulación sugiere también una forma distinta de codificar. Supongamos que $f_0, f_1, ..., f_{k-1}$ son k son los símbolos de información y $f(x) = f_0 + f_1 x + ... + f_{k-1}x^{k-1}$, entonces

$$(f_0, f_1, ..., f_{k-1}) \xrightarrow{\text{codificar}} (f(1), f(\alpha), ..., f(\alpha^{q-2}))$$

Notamos que este sistema de codificación no es sistemático.

3.4. Códigos Reed-Solomon Generalizados

Finalmente llegamos a los códigos que ponen nombre a esta sección y que van a ser los más importantes para nuestro estudio.

Definición 3.4.1 (Códigos RS Generalizados)**.** Sean $v_1, ..., v_n \in \mathbb{F}_q$ distintos de cero y $\alpha_1, ..., \alpha_n \in \mathbb{F}_q$ diferentes entre sí. Fijemos $v = (v_1, ...v_n)$ y $\alpha = (\alpha_1, ..., \alpha_n)$. Para $0 \leq k \leq n$ definimos los códigos RS generalizados como

$$GRS_{n,k}(\alpha, v) = \{(v_1 f(\alpha_1), v_2 f(\alpha_2), ..., v_n f(\alpha_n)) | f(x) \in \mathcal{P}_k\}.$$

En lo que sigue utilizaremos la notación $ev_{\alpha,v}(f(x)) = (v_1 f(\alpha_1), v_2 f(\alpha_2), ..., v_n f(\alpha_n))$, indicando que la palabra código $f = ev_{\alpha,v}(f(x))$ surge de evaluar el polinomio $f(x)$ en α escalado por v. En el siguiente teorema vemos que los códigos GRS (por sus siglas en inglés) pertenecen a una de las familias ya presentadas.

Teorema 3.4.1. *$GRS_{n,k}(\alpha, v)$ es un $[n,k]_q$-código lineal con longitud $n \leq q$. Tenemos que la distancia mínima d_{min} es igual a $n - k + 1$ supuesto que $k \neq 0$. En particular, los códigos GRS son MDS.*

Demostración. Por definición, las entradas en α son diferentes, luego tenemos que $n \leq q$. Si $a, b \in \mathbb{F}_q$ y $f(x), g(x) \in \mathcal{P}_k$, entonces $af(x) + bg(x) \in \mathcal{P}_k$ y

$$ev_{\alpha,v}(af(x) + bg(x)) = aev_{\alpha,v}(f(x)) + bev_{\alpha,v}(g(x)) = af + bg.$$

Por tanto $GRS_{n,k}(\alpha, v)$ es lineal de longitud n sobre $\mathbb{F}_q$.
Sean $f(x), g(x) \in \mathcal{P}_k$ polinomios diferentes. Sea $h(x) = f(x) - g(x) \neq 0$, con gr $h \leq k$. Entonces $h = f - g$ y $wt(h) = d(f, g)$. Pero el peso de h es n menos el número de 0s en h. Como todos los v_i son distintos de cero, este es igual a n menos el número de raíces que $h(x)$ tenga en $\{\alpha_1, ..., \alpha_n\}$. Como $h(x)$ tiene como mucho $k - 1$ raíces, el peso de h es al menos $n - (k - 1) = n - k + 1$. En consecuencia $d_{min} \geq n - k + 1$, y obtenemos la igualdad de la cota de Singleton.
El argumento del párrafo anterior demuestra también que polinomios distintos $f(x), g(x) \in \mathcal{P}_k$ dan palabras código diferentes. Por lo tanto, el código contiene q^k palabras código y tiene dimensión k. □

Ejemplo 3.4.1. Sean $v = (4, 5, 3, 2, 4, 5, 6, 1, 2)$, con $v_i \in \mathbb{F}_{13}, v_i \neq 0, \forall i \in \{1, .., 9\}$ y $\alpha = (1, 4, 3, 2, 6, 9, 10, 11, 7)$, con $\alpha_i \in \mathbb{F}_{13}, \alpha_i \neq \alpha_j, \forall i, j \in \{1, .., 9\}, i \neq j$. Podemos definir por tanto el código RS generalizado $GRS_{9,4}(\alpha, v)$. Si escogemos $f(x) = 2x + 4 \in \mathcal{P}_4$, tenemos que $(4f(1), 5f(4), 3f(3), 2f(2), 4f(6), 5f(9), 6f(10), f(11), 2f(7)) =$
$(11, 8, 4, 3, 12, 6, 1, 0, 10) \in GRS_{9,4}(\alpha, v)$. Además, la distancia mínima de este código es $9 - 4 + 1 = 6$ por el teorema 3.4.1.

Encontremos ahora una matriz generadora para $GRS_{n,k}(\alpha, v)$. El argumento del teorema 3.4.1 deja claro que cualquier base $f_1(x), ..., f_k(x)$ de $\mathcal{P}_k$ da lugar a una base $f_1, ..., f_k$ del código. Escojamos la base usual de $\mathcal{P}_k$, es decir, el conjunto $\{1, x, .., x^i, ..., x^{k-1}\}$. La matriz generadora asociada, cuya i-ésima fila es

$$ev_{\alpha,v}(x^i) = (v_1\alpha_1^i, ..., v_j\alpha_j^i, ..., v_n\alpha_n^i)$$

constituye una matriz generadora para $GRS_{n,k}(\alpha, v)$

$$\begin{pmatrix} v_1 & v_2 & \dots & v_j & \dots & v_n \\ v_1\alpha_1 & v_2\alpha_2 & \dots & v_j\alpha_j & \dots & v_n\alpha_n \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ v_1\alpha_1^i & v_2\alpha_2^i & \dots & v_j\alpha_j^i & \dots & v_n\alpha_n^i \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ v_1\alpha_1^{k-1} & v_2\alpha_2^{k-1} & \dots & v_j\alpha_j^{k-1} & \dots & v_n\alpha_n^{k-1} \end{pmatrix}. \tag{3.3}$$

Puede resultar útil pensar en el teorema 3.4.1 en términos de interpolación polinómica. Recordemos que cualquier polinomio de grado menor que k está únicamente determinado por sus valores en k o más puntos distintos.
Dada cualquier n-tupla f podemos fácilmente reconstruir el único polinomio $f(x)$ de grado menor o igual que k con $f = ev_{\alpha,v}(f(x))$. Sean $L(x) = \prod_{i=1}^{n}(x - \alpha_i)$ y
$L_i(x) = \frac{L(x)}{x-\alpha_i} = \prod_{j\neq i}(x - \alpha_j)$.
Los polinomios $L(x)$ y $L_i(x)$ son de grado n y $n-1$, respectivamente. Usando la fórmula de interpolación de Lagrange

$$f(x) = \sum_{i=1}^{n} \frac{L_i(x)}{L_i(\alpha_i)} f(\alpha_i).$$

Veamos una aplicación de este concepto en la demostración del siguiente resultado.

Teorema 3.4.2. *Tenemos que*

$$GRS_{n,k}(\alpha, v)^{\perp} = GRS_{n,n-k}(\alpha, u),$$

donde $u = (u_1, ..., u_n)$ *con* $u_i^{-1} = v_i \prod_{j\neq i}(\alpha_i - \alpha_j)$.

Demostración. Por definición $u_i = v_i^{-1} L_i(\alpha_i)^{-1}$. Vamos a probar que un f en $GRS_{n,k}(\alpha, v)$ es ortogonal con cualquier g en $GRS_{n,n-k}(\alpha, u)$ demostrando que su producto escalar es 0. Sea $f = ev_{\alpha,v}(f(x))$ y $g = ev_{\alpha,u}(g(x))$. Tenemos que gr $f(x) \leq k$ mientras que gr $g(x) \leq n - k$. Por tanto su producto $f(x)g(x)$ tiene grado como mucho $n-2$. Por la interpolación de Lagrange

$$f(x)g(x) = \sum_{i=1}^{n} \frac{L_i(x)}{L_i(\alpha_i)} f(\alpha_i)g(\alpha_i).$$

Igualando el coeficiente de x^{n-1} en ambos lados nos da

$$0 = \sum_{i=1}^{n} \frac{1}{L_i(\alpha_i)} f(\alpha_i)g(\alpha_i) = \sum_{i=1}^{n} (v_i f(\alpha_i))(\frac{v_i^{-1}}{L_i(\alpha_i)} g(\alpha_i)) = \sum_{i=1}^{n} (v_i f(\alpha_i))(u_i g(\alpha_i)) = f \cdot g,$$

que es lo que buscábamos. □

Ejemplo 3.4.2. Sea $GRS_{4,2}(\alpha, v)$ sobre $\mathbb{F}_7$, con $\alpha = (1,2,3,4), v = (3,6,6,2)$. Calculemos el vector u tal y como se describe en el enunciado del teorema 3.4.2, por lo que hemos de calcular las inversas de sus componentes:

$$\begin{aligned}
u_1^{-1} &= v_1 \cdot (\alpha_1 - \alpha_2) \cdot (\alpha_1 - \alpha_3) \cdot (\alpha_1 - \alpha_4) = 3 \cdot 6 \cdot 5 \cdot 3 = 4 \bmod 7, \\
u_2^{-1} &= v_2 \cdot (\alpha_2 - \alpha_1) \cdot (\alpha_2 - \alpha_3) \cdot (\alpha_2 - \alpha_4) = 6 \cdot 1 \cdot 6 \cdot 5 = 5 \bmod 7, \\
u_3^{-1} &= v_3 \cdot (\alpha_3 - \alpha_1) \cdot (\alpha_3 - \alpha_2) \cdot (\alpha_3 - \alpha_4) = 6 \cdot 2 \cdot 1 \cdot 6 = 2 \bmod 7, \\
u_4^{-1} &= v_4 \cdot (\alpha_4 - \alpha_1) \cdot (\alpha_4 - \alpha_2) \cdot (\alpha_4 - \alpha_3) = 2 \cdot 3 \cdot 2 \cdot 1 = 5 \bmod 7.
\end{aligned}$$

Nos queda $u = (2,3,4,3)$, por lo que, de acuerdo con el teorema 3.4.2, $GRS_{4,2}(\alpha, v)^{\perp} = GRS_{4,2}(\alpha, u)$.

Podemos caracterizar por tanto los elementos de $C = GRS_{n,k}(\alpha, v)$ en términos de su ortogonalidad con las palabras código de $C^{\perp} = GRS_{n,n-k}(\alpha, u)$. Sin embargo, no tenemos que comprobarlo con todas las palabras de $C^{\perp}$, sino que basta con una base suya, por ejemplo la usual. Sea $r = n - k$, y $c = (c_1, ..., c_n) \in \mathbb{F}_q^n$. Entonces

$$c \in C \Leftrightarrow 0 = c \cdot ev_{\alpha,u}(x^j), \text{ para } 0 \leq j \leq r-1 \Leftrightarrow 0 = \sum_{i=1}^{n} c_i u_i \alpha_i^j, \text{ para } 0 \leq j \leq r-1.$$

Reescribimos estas r ecuaciones como una sola ecuación en el anillo de polinomios $\mathbb{F}_q[z]$ en una nueva indeterminada z. El vector c está en C si y solo si en $\mathbb{F}_q[z]$ se da

$$0 = \sum_{j=0}^{r-1} (\sum_{i=1}^{n} c_i u_i \alpha_i^j) z^j = \sum_{i=1}^{n} c_i u_i (\sum_{j=0}^{r-1} (\alpha_i z)^j).$$

Tenemos que $mcd(1 - \alpha z, z^r) = 1$, por lo que $1 - \alpha z$ es invertible en el anillo $\frac{\mathbb{F}_q[z]}{z^r}$. De hecho

$$\frac{1}{1 - \alpha z} \equiv \sum_{j=0}^{r-1} (\alpha z)^j \pmod{z^r},$$

que es un truncamiento de la expansión usual de la serie geométrica. Llegamos al siguiente resultado.

Teorema 3.4.3 (Formulación de Goppa para los códigos GRS). *El código RS generalizado $GRS_{n,k}(\alpha, v)$ sobre $\mathbb{F}_q$ es igual al conjunto de todas las n-tuplas $c = (c_1, c_2, ..., c_n) \in \mathbb{F}_q^n$ tales que*

$$\sum_{i=1}^{n} \frac{c_i u_i}{1 - \alpha_i z} \equiv 0 \ (mod\ z^r)\ ,$$

donde $r = n - k$ y u_i como en el teorema 3.4.2

Esta interpretación de los códigos GRS sugiere un método práctico para su decodificación. Supongamos que $c = (c_1, c_2, ..., c_n) \in GRS_{n,k}(\alpha, v)$ es transmitido e $y = (y_1, y_2, .., y_n)$ es recibido, habiéndose introducido el vector de error $e = (e_1, e_2, ...e_n)$, de manera que

$y = c + e$. Definimos el polinomio síndrome de y como:

$$S_y(z) \equiv \sum_{i=1}^{n} \frac{y_i u_i}{1 - \alpha_i z} \pmod{z^r}.$$

Vemos fácilmente que

$$S_y(z) \equiv S_c(z) + S_e(z) \pmod{z^r}$$

y usando la formulación de Goppa del teorema 3.4.3

$$S_y(z) \equiv S_e(z) \pmod{z^r}.$$

Sea B el conjunto de los índices de e con entrada no nula, es decir, el conjunto de las localizaciones de los errores:

$$B = \{i | e_i \neq 0\}.$$

Entonces tenemos que

$$S_y(z) \equiv S_e(z) \equiv \sum_{b \in B} \frac{e_b u_b}{1 - \alpha_b z} \pmod{z^r}.$$

Eliminemos los subíndices y denotemos simplemente por $S(z)$ al polinomio síndrome de y. Si quitamos denominadores obtenemos la denominada ecuación clave:

$$\sigma(z)S(z) \equiv \omega(z) \pmod{z^r},$$

donde

$$\sigma(z) = \sigma_e(z) = \prod_{b \in B}(1 - \alpha_b z)$$

y

$$\omega(z) = \omega_e(z) = \sum_{b \in B} e_b u_b \Big(\prod_{a \in B, a \neq b} (1 - \alpha_a z) \Big).$$

El polinomio $\sigma(z)$ es llamado el polinomio localizador de errores, y el polinomio $\omega(z)$ es el evaluador de errores.
Dados los polinomios $\sigma(z) = \sigma_e(z)$ y $\omega(z) = \omega_e(z)$, podemos reconstruir el vector de error e. Supongamos por el momento que ninguno de los α_i es igual a 0. Entonces:

$$B = \{b | \sigma(\alpha_b^{-1}) = 0\}$$

y, para cada $b \in B$

$$e_b = \frac{-\alpha_b \omega(\alpha_b^{-1})}{u_b \sigma'(\alpha_b^{-1})}, \tag{3.4}$$

donde $\sigma'(z)$ es la derivada de $\sigma(z)$. De hecho los polinomios $\sigma(z)$ y $\omega(z)$ determinan el vector de error incluso cuando algún α_i es 0. En la siguiente proposición se recogen algunas propiedades de los dos polinomios recién introducidos.

Proposición 3.4.4. *Sean $\sigma(z)$ y $\omega(z)$ los polinomios localizador de errores y de evaluación para el vector de error $e \neq 0$. Fijemos $B = \{b | e_b \neq 0\}$ y $B_0 = \{b \in B | \alpha_b \neq 0\}$. Supongamos que hay como mucho un índice b con $\alpha_b = 0$. Tenemos lo siguiente:*

1. *$wt(e) = |B|$ es igual a $|B_0| + 1$ o $|B_0|$ dependiendo si hay o no un índice b con $\alpha_b = 0$ y $e_b \neq 0$.*

2. *gr* $\sigma(z) = |B_0|$ *y gr* $\omega(z) \leq |B| - 1$.

3. $B_0 = \{b | \alpha_b \neq 0, \sigma(\alpha_b^{-1}) = 0\}$.

4. *El índice b con* $\alpha_b = 0$ *pertenece a B si y solo si gr* $\sigma(z)$ = *gr* $\omega(z)$.

5. *Para* $b \in B_0, e_b$ *es dado por la fórmula (3.4). Si* $b \in B \setminus B_0$ *entonces*

$$e_b = wu_b^{-1}(\prod_{a \in B_0}(-\alpha_a))^{-1},$$

donde w es el coeficiente líder de $\omega(z)$.

Si el vector de error $e \neq 0$ tiene peso como mucho $\frac{r}{2}(= \frac{d_{min}-1}{2})$, entonces en relación con el polinomio síndrome $S_e(z) = S(z)$ el par de polinomios $\sigma_e(z) = \sigma(z)$ y $\omega_e(z) = \omega(z)$ tiene tres propiedades por las cuales son caracterizados en el siguiente teorema. La propiedad 1) se reduce a la ecuación clave. La propiedad 2) es una consecuencia de la hipotésis sobre el peso del error y las definiciones de los polinomios $\sigma(z)$ y $\omega(z)$ hechas anteriormente. Para 3) tenemos que $\sigma(0) = 1$ trivialmente. Como $\sigma(z)$ tiene gr $\sigma(z)$ raíces distintas, o bien $mcd(\sigma(z), \omega(z)) = 1$ o los dos polinomios tienen una raíz común. Pero por cada raíz α_b^{-1} de $\sigma(z)$ tenemos que $0 \neq \omega(\alpha_b^{-1})$, un factor de $e_b \neq 0$.
Nuestro objetivo va a ser, por tanto, resolver la ecuación clave, encontrando así el vector de error e. El siguiente teorema nos da una caracterización de las soluciones que estamos buscando.

Teorema 3.4.5. *Dados r y* $S(z) \in \mathbb{F}_q[z]$ *hay como mucho un par de polinomios* $\sigma(z), \omega(z)$ *en* $\mathbb{F}_q[z]$ *satisfaciendo:*

1. $\sigma(z)S(z) \equiv \omega(z)$ *(mod* z^r*).*

2. *gr* $(\sigma(z)) \leq \frac{r}{2}$ *y gr* $(\omega(z)) < \frac{r}{2}$.

3. $mcd(\sigma(z), \omega(z)) = 1$ *y* $\sigma(0) = 1$.

De hecho, podemos demostrar algo ligeramente más fuerte si relajamos la tercera propiedad.

Proposición 3.4.6. *Supongamos que* $\sigma(z), \omega(z)$ *satisfacen los puntos* $1) - 3)$ *del Teorema 3.4.5 y que* $\sigma_1(z), \omega_1(z)$ *satisfacen* $1)$ *y* $2)$. *Entonces existe un polinomio* $\mu(z)$ *con* $\sigma_1(z) = \mu(z)\sigma(z)$ *y* $\omega_1(z) = \mu(z)\omega(z)$.

Demostración. Aplicando 1) a ω_1 y σ obtenemos que

$$\sigma(z)\omega_1(z) = \sigma(z)\sigma_1(z)S(z) \equiv \sigma_1(z)\omega(z) \pmod{z^r}.$$

Por tanto

$$\sigma(z)\omega_1(z) - \sigma_1(z)\omega(z) \equiv 0 \pmod{z^r}.$$

Pero por 2), el lado izquierdo de esta ecuación tiene grado menor o igual que r. En consecuencia

$$\sigma(z)\omega_1(z) = \sigma_1(z)\omega(z).$$

De 3) tenemos que $mcd(\sigma(z), \omega(z)) = 1$, por lo que necesariamente $\sigma(z)$ divide a $\sigma_1(z)$, existiendo $\mu(z) \in \mathbb{F}_q[z]$ tal que $\sigma_1(z) = \mu(z)\sigma(z)$. Entonces

$$\sigma(z)\omega_1(z) = \sigma_1(z)\omega(z) = \sigma(z)\mu(z)\omega(z).$$

El polinomio $\sigma(z)$ no es cero ya que $\sigma(0) = 1$ por 3), por lo que por cancelación, $\omega_1(z) = \mu(z)\omega(z)$, como buscábamos. □

Podemos demostrar ahora la unicidad en el teorema 3.4.5.

Demostración. Si existiera otro tal par verificaría

$$\sigma_1(z) = \mu(z)\sigma(z) \text{ y } \omega_1(z) = \mu(z)\omega(z),$$

por la proposición anterior. Por lo tanto, $\mu(z)$ divide a $mcd(\sigma_1(z), \omega_1(z))$, el cual es 1 por 3). En consecuencia, $\mu(z) = \mu$ es una constante. De hecho

$$1 = \sigma_1(0) = \mu(0)\sigma(0) = \mu \cdot 1 = \mu.$$

Finalmente $\sigma_1(z) = \mu(z)\sigma(z) = \sigma(z)$ y $\omega_1(z) = \mu(z)\omega(z) = \omega(z)$. □

Usando la caracterización del teorema 3.4.5 presentamos ahora un método para resolver la ecuación clave con el algoritmo de Euclides.

Teorema 3.4.7. *(Decodificación de GRS usando el Algoritmo de Euclides) Consideremos el código $GRS_{n,k}(\alpha, v)$ sobre $\mathbb{F}_q$, y fijemos $r = n - k$. Dado un polinomio síndrome $S(z)$, el siguiente algoritmo finaliza, produciendo polinomios $\tilde{\sigma}(z)$ y $\tilde{\omega}(z)$:*

Fijamos z^r y $S(z)$ como entradas
Mientras gr $(r_j(z)) \geq \frac{r}{2}$ **hacer**
  Ejecutamos el algoritmo de Euclides
Terminar Mientras
$\tilde{\sigma}(z) = t_j(z)$
$\tilde{\omega}(z) = r_j(z)$

Donde $t_j(z)$ y $r_j(z)$ son los correspondientes valores del algoritmo de Euclides del teorema 1.2.4 en el paso j. Si hay una palabra de error e de peso como mucho $\frac{r}{2} = \frac{d_{min}-1}{2}$ con $S_e(z) = S(z)$, entonces $\hat{\sigma}(z) = \tilde{\sigma}(0)^{-1}\tilde{\sigma}(z)$ y $\hat{\omega}(z) = \tilde{\sigma}(0)^{-1}\tilde{\omega}(z)$ son los polinomios localizador de errores y evaluador para e.

Demostración. El fin del algoritmo de Euclides es disminuir el grado de $r_j(z)$ en cada paso, por lo que está garantizado que el algoritmo pare.
Supongamos ahora que $S(z) = S_e(z)$ con $wt(e) \leq \frac{r}{2}$, de manera que los polinomios localizador y evaluador existen, $\sigma(z) = \sigma_e(z)$ y $\omega(z) = \omega_e(z)$, satisfaciendo los puntos $1) - 3)$ del teorema 3.4.5. Comprobemos que, dado el j para el cual el algoritmo sale del bucle, el par $t_j(z)$ y $r_j(z)$ satisfacen 1) y 2).
El requisito 1) es simplemente la ecuación clave y es satisfecha en cada paso del algoritmo de Euclides ya que siempre

$$E_j : r_j(z) = s_j(z)z^r + t_j(z)S(z) \equiv t_j(z)S(z) \pmod{z^r}.$$

Además gr $r_j(z) < \frac{r}{2}$ y gr $r_{j-1}(z) \geq \frac{r}{2}$. Por tanto,

$$\text{gr } t_j(z) + \frac{r}{2} \leq \text{gr } t_j(z) + \text{gr } r_{j-1}(z) = \text{gr } z^r = r.$$

En consecuencia, gr $t_j(z) \leq \frac{r}{2}$, dándonos 2). Por la proposición 3.4.6 existe un polinomio $\mu(z)$ no nulo con

$$t_j(z) = \mu(z)\sigma(z) \text{ y } r_j(z) = \mu(z)\omega(z).$$

Si sustituimos $t_j(z)$ y $r_j(z)$ en la ecuación E_j tenemos que

$$s_j(z)z^r + (\mu(z)\sigma(z))S(z) = \mu(z)\omega(z),$$

el cual se vuelve

$$\mu(z)(\omega(z) - \sigma(z)S(z)) = s_j(z)z^r.$$

Por la ecuación clave, la expresión entre paréntesis de la izquierda es $p(z)z^r$, para algún $p(z) \in \mathbb{F}_q[z]$, por lo que nos quedamos con $\mu(z)p(z)z^r = s_j(z)z^r$, y despejando z^r, $\mu(z)p(z) = s_j(z)$ Por tanto, $\mu(z)$ divide a $mcd(t_j(z), s_j(z))$, el cual es 1.
Concluimos que $\mu(z) = \mu$ es una función constante distinta de cero. Además, usando que $\sigma(z)$ verifica 3)

$$t_j(0) = \mu(0)\sigma(0) = \mu,$$

por lo que

$$\sigma(z) = t_j(0)^{-1}t_j(z) \text{ y } \omega(z) = t_j(0)^{-1}r_j(z),$$

como buscábamos. □

Si recordamos la expresión de las componentes del vector de error que hemos derivado en 3.4, la decodificación no se produce cuando $\hat{\sigma}(z)$ no se descompone en factores lineales cuyas raíces son inversas de entradas en α con multiplicidad 1 (aquí asumimos que ninguno de los α_i son 0), pues si la multiplicidad es mayor, llegaríamos a un denominador nulo. Es decir, el número de raíces de $\hat{\sigma}(z)$ entre los α_i^{-1} debe ser igual al grado de $\hat{\sigma}(z)$. Si este no es el caso, entonces hemos detectado errores que no somos capaces de corregir. Tampoco podríamos decodificar cuando $t_j(0) = 0$, ya que la división final para determinar $\hat{\sigma}(z)$ no puede ser realizada.
Si $t_j(0) \neq 0$ y $\hat{\sigma}(z)$ factoriza como hemos descrito, entonces podemos proceder a evaluar errores en cada una de las posiciones localizadas y encontrar un vector de error, de peso como mucho $\frac{r}{2}$, con nuestro polinomio síndrome original. En este caso, o bien hemos decodificado correctamente, o bien se han producido más de $\frac{r}{2}$ errores y la decodificación no ha tenido éxito.
Supongamos ahora que r es par o que α tiene peso n. Entonces este algoritmo solo produce vectores de error de peso $\frac{r}{2}$ o menos. En particular, si ocurren más de $\frac{r}{2}$ errores entonces tendremos una falta de decodificado o un error de decodificado. Supongamos que hemos encontrado polinomios $\hat{\sigma}(z)$ y $\hat{\omega}(z)$ que nos permiten calcular un candidato a vector de error e de peso como mucho $\frac{r}{2}$. Se sigue de la interpolación de Lagrange que $\hat{\sigma}(z) = \sigma_e(z)$ y $\hat{\omega}(z) = \omega_e(z)$. También, ya que $\sigma(z)$ es invertible módulo z^r, podemos resolver la ecuación clave para encontrar que $S(z) = S_e(z)$. Por tanto, el vector recibido está dentro de una esfera de radio $\frac{r}{2}$ alrededor de una palabra código y es decodificado a dicha palabra, aplicando el corolario 2.2.1. Veamos un ejemplo práctico de cálculo del vector de error.

Ejemplo 3.4.3. Consideremos el código $C = GRS_{6,2}(\alpha, v)$ sobre el cuerpo $\mathbb{F}_{11}$ de los enteros

módulo 11, donde

$$\alpha = (3, 2, 7, 1, 4, 5)$$

y

$$v = (2, 2, 2, 2, 2, 2).$$

Vamos a trabajar con $C^{\perp} = GRS_{6,4}(\alpha, u)$, por lo que debemos calcular el vector u. Empezando con

$$L(x) = (x-3)(x-2)(x-7)(x-1)(x-4)(x-5)$$

encontramos:

$$\begin{aligned}
L_1(3) &= (1)(-4)(2)(-1)(-2) = -16 \equiv 6 \bmod 11,\\
L_2(2) &= (-1)(-5)(1)(-2)(-3) = 30 \equiv 8 \bmod 11,\\
L_3(7) &= (4)(5)(6)(3)(2) = 720 \equiv 5 \bmod 11,\\
L_4(1) &= (-2)(-1)(-6)(-3)(-4) = -144 \equiv 10 \bmod 11,\\
L_5(4) &= (1)(2)(-3)(3)(-1) = 18 \equiv 7 \bmod 11,\\
L_6(5) &= (2)(3)(-2)(4)(1) = -48 \equiv 7 \bmod 11.
\end{aligned}$$

Ahora $u_i = (v_i L_i(\alpha_i))^{-1}$:

$$\begin{aligned}
u_1 &= (2 \cdot L_1(3))^{-1} \bmod 11 \equiv (2 \cdot 6)^{-1} \bmod 11 \equiv (1)^{-1} \bmod 11 \equiv 1 \bmod 11,\\
u_2 &= (2 \cdot L_2(2))^{-1} \bmod 11 \equiv (2 \cdot 8)^{-1} \bmod 11 \equiv (5)^{-1} \bmod 11 \equiv 9 \bmod 11,\\
u_3 &= (2 \cdot L_3(7))^{-1} \bmod 11 \equiv (2 \cdot 5)^{-1} \bmod 11 \equiv (10)^{-1} \bmod 11 \equiv 10 \bmod 11,\\
u_4 &= (2 \cdot L_4(1))^{-1} \bmod 11 \equiv (2 \cdot 10)^{-1} \bmod 11 \equiv (9)^{-1} \bmod 11 \equiv 5 \bmod 11,\\
u_5 &= (2 \cdot L_5(4))^{-1} \bmod 11 \equiv (2 \cdot 7)^{-1} \bmod 11 \equiv (3)^{-1} \bmod 11 \equiv 4 \bmod 11,\\
u_6 &= (2 \cdot L_6(5))^{-1} \bmod 11 \equiv (2 \cdot 7)^{-1} \bmod 11 \equiv (3)^{-1} \bmod 11 \equiv 4 \bmod 11.
\end{aligned}$$

Por lo que

$$u = (1, 9, 10, 5, 4, 4).$$

A continuación calculamos el polinomio síndrome de un vector arbitrario recibido

$$y = (y_1, y_2, y_3, y_4, y_5, y_6).$$

En nuestro ejemplo $r = 6 - 2 = 4$.

$$\begin{aligned}
S_y(z) &\equiv \frac{1 \cdot y_1}{1-3z} + \frac{9 \cdot y_2}{1-2z} + \frac{10 \cdot y_3}{1-7z} + \frac{5 \cdot y_4}{1-z} + \frac{4 \cdot y_5}{1-4z} + \frac{4 \cdot y_6}{1-5z} \pmod{z^4}\\
&\equiv y_1(1+3z+9z^2+5z^3) + 9y_2(1+2z+4z^2+8z^3) + 10y_3(1+7z+5z^2+2z^3) + 5y_4(1+z+z^2+z^3)\\
&\quad + 4y_5(1+4z+5z^2+9z^3) + 4y_6(1+5z+3z^2+4z^3) \pmod{z^4}\\
&\equiv y_1(1+3z+9z^2+5z^3) + y_2(9+7z+3z^2+6z^3) + y_3(10+4z+6z^2+9z^3)\\
&\quad + y_4(5+5z+5z^2+5z^3) + y_5(4+5z+9z^2+3z^3)\\
&\quad + y_6(4+9z+z^2+5z^3) \pmod{z^4}.
\end{aligned}$$

Usamos ahora el algoritmo del teorema 3.4.7 para decodificar el vector recibido

$$y = (5, 10, 8, 6, 10, 0).$$

Tenemos el polinomio síndrome

$$\begin{aligned} S(z) &\equiv \frac{1\cdot 5}{1-3z} + \frac{9\cdot 10}{1-2z} + \frac{10\cdot 8}{1-7z} + \frac{5\cdot 6}{1-z} + \frac{4\cdot 10}{1-4z} + \frac{4\cdot 0}{1-5z} \pmod{z^4} \\ &\equiv 5(1+3z+9z^2+5z^3) \\ &+ 10(9+7z+3z^2+6z^3) \\ &+ 8(10+4z+6z^2+9z^3) \\ &+ 6(5+5z+5z^2+5z^3) \\ &+ 10(4+5z+9z^2+3z^3) \pmod{z^4} \\ &\equiv 3+10z+z^2+8z^3 \pmod{z^4}. \end{aligned}$$

El algoritmo requiere ahora que, empezando con las condiciones iniciales

$$a(z) = z^4 \text{ y } b(z) = 8z^3 + z^2 + 10z + 3,$$

ejecutemos el algoritmo de Euclides hasta que en el paso j tengamos por primera vez que gr $r_j(z) < \frac{r}{2} = 2$.
En el paso 2 tenemos la primera aparición de un resto con grado menor que 2; tenemos que $r_2(z) = 3$. Obtenemos también que $t_2(z) = z^2 + 4z + 1$, por lo que $t_2(0)^{-1} = 1^{-1} \equiv 1 \bmod 11$.
Por lo tanto, nos quedan como polinomios localizador y evaluador de errores:

$$\sigma(z) = t_2(0)^{-1} t_2(z) = 1(z^2 + 4z + 1) \equiv z^2 + 4z + 1 \bmod 11,$$
$$\omega(z) = t_2(0)^{-1} r_2(z) \equiv 3 \bmod 11.$$

Las localizaciones de los errores son aquellas en $B = \{b | \sigma(\alpha_b^{-1}) = 0\}$. En primer lugar, vamos a calcular $\alpha_i^{-1}, \forall i \in \{1, 2, 3, 4, 5, 6\}$:

$$\begin{aligned} \alpha_1^{-1} &= 3^{-1} \equiv 4 \bmod 11, \\ \alpha_2^{-1} &= 2^{-1} \equiv 6 \bmod 11, \\ \alpha_3^{-1} &= 7^{-1} \equiv 8 \bmod 11, \\ \alpha_4^{-1} &= 1^{-1} \equiv 1 \bmod 11, \\ \alpha_5^{-1} &= 4^{-1} \equiv 3 \bmod 11, \\ \alpha_6^{-1} &= 5^{-1} \equiv 9 \bmod 11. \end{aligned}$$

Evaluamos ahora σ en cada uno de los α_i^{-1}:

$$\sigma(\alpha_1^{-1}) \equiv 0 \bmod 11,$$
$$\sigma(\alpha_2^{-1}) \equiv 6 \bmod 11,$$
$$\sigma(\alpha_3^{-1}) \equiv 9 \bmod 11,$$
$$\sigma(\alpha_4^{-1}) \equiv 6 \bmod 11,$$
$$\sigma(\alpha_5^{-1}) \equiv 0 \bmod 11.$$

Tenemos que α_1^{-1} y α_5^{-1} son raíces de σ, por lo que $B = \{1, 5\}$.
Un valor de error e_b es dado por

$$e_b = \frac{-\alpha_b \omega(\alpha_b^{-1})}{u_b \sigma'(\alpha_b^{-1})},$$

donde $\sigma'(z) = 2z + 4$. Entonces $e_1 = \frac{-3 \cdot 3}{1 \cdot 12} \equiv 2 \bmod 11$ y $e_5 = \frac{-4 \cdot 3}{4 \cdot 10} \equiv 3 \bmod 11$.
Hemos encontrado, por tanto,

$$e = (2, 0, 0, 0, 3, 0)$$

y decodificamos la palabra recibida y a

$$c = y - e = (5, 10, 8, 6, 10, 0) - (2, 0, 0, 0, 3, 0) = (3, 10, 8, 6, 7, 0).$$

En el ejemplo, hemos utilizado el algoritmo extendido de Euclides para obtener el polinomio localizador de errores σ. Si, como en nuestro caso, las raíces de σ se encuentran entre los α_i^{-1}, basta con sustituir todos estos elementos para conocer la posición de los errores, sin ser necesario realizar la evaluación en todos los elementos del cuerpo. En consecuencia, serían necesarias n evaluaciones (6 en nuestro caso), siendo el número de evaluaciones independiente del tamaño del cuerpo.

Como fin de esta sección, vamos a presentar una posible implementación en Python de un codificador-decodificador usando códigos RS Generalizados, que podemos encontrar en el archivo *encDecGRS.py*. Necesitaremos tener instalados en nuestro equipo tanto Python como SageMath para poder ejecutarlo. Admite hasta 11 parámetros para su ejecución, debiéndose pasar todos ellos con – como prefijo (por ejemplo –longitud). En primer lugar, *longitud*, que se refiere a la longitud del código GRS que se desea utilizar, debe ser un entero y es obligatorio introducirlo. En segundo lugar, *ordenCuerpo*, que se refiere al tamaño del cuerpo sobre el que vamos a construir el código, debe ser un entero mayor o igual que *longitud* y es obligatorio. En tercer lugar, *polinomioBase*, que se refiere al polinomio irreducible necesario para construir los cuerpos que no son de orden primo, a partir de uno que sí. Consiste en una lista de enteros (todos los enteros pasados como argumento se representarán módulo el cuerpo o subcuerpo primo en cuestión, aunque sean negativos o mayores o iguales que el orden del cuerpo o subcuerpo) que se pasan separados por espacio, representando los coeficientes de un polinomio de grado el número de elementos pasados menos uno. Por ejemplo, si se pasa 1 2 3 4, el programa interpretaría el polinomio $4x^3 + 3x^2 + 2x + 1$. No es obligatorio, solo se tiene que utilizar en los casos en que *ordenCuerpo* no sea primo. En cuarto lugar, *ordenSubcuerpoPrimo*, que utilizaremos junto a *polinomioBase* para construir los cuerpos no primos. Debe ser un entero primo y no es obligatorio, igual que *polinomioBase*. En quinto lugar, *dimension*, que representa la

dimensión del código a utilizar, debe ser un entero mayor o igual que 0, menor o igual que *longitud* y es obligatorio. En sexto lugar, *alpha*, que representa al vector α que define un código GRS. En el caso de que estemos trabajando con un cuerpo primo, pasaremos simplemente una lista, de tamaño *longitud* sin repeticiones, de enteros pertenecientes al cuerpo separados por espacio. Por ejemplo, $\alpha = (1, 2, 3, 4)$ en $\mathbb{F}_5$ se pasaría como 1 2 3 4. Sin embargo, si estamos en el caso no primo, tenemos que expresar cada elemento de la lista en términos de una base del cuerpo, por lo que pasaremos coeficientes en dicha base separados por comas. Sea $\mathbb{F}_q$, con $q = p^n$, $\mathbb{F}_q \cong \frac{\mathbb{F}_p}{f(x)}$, con $f(x) \in \mathbb{F}_p[x]$ irreducible de grado n, y a una raíz de $f(x)$ en $\mathbb{F}_q$, tendremos que una base de $\mathbb{F}_q$ sobre $\mathbb{F}_p$ vendrá dada por $\{1, a, ..., a^{n-1}\}$. Por ejemplo, si estamos en $\mathbb{F}_{25}$ con polinomio irreducible $x^2 + x + 1 \in \mathbb{F}_5[x]$, la base será $\{1, a\}$ y si queremos enviar $\alpha = (2a + 1, 3, 4a, a + 2)$ tendremos que pasar como argumento 2, 1 3 4, 0 1, 2, es decir, vamos de mayor a menor grado de a. Es obligatorio introducirlo. En séptimo lugar, v, que representa al vector v que define un código GRS. No puede haber 0s entre sus componentes, debe tener tamaño *longitud*, es obligatorio introducirlo y sigue el mismo formato que el argumento *alpha*. En octavo lugar, *modo*, que es una cadena de texto que representa los cuatro modos de ejecución disponibles: *codificar* (transformar un mensaje en una palabra del código usado), *decodificar* (convertir una palabra código en el mensaje de la que proviene), *introducirErrores* (añadir un vector de errores a una palabra código) y *corregirCodigo* (localizar las posiciones donde pudieran haberse añadido errores y devolver la palabra código sin ellos). Es obligatorio introducirla. En noveno lugar, *mensaje*, que solamente utilizaremos junto con el modo *codificar*, debe tener tamaño *dimension* y sigue el mismo formato que el argumento *alpha*. En décimo lugar, *palabraCodigo*, que utilizaremos junto con los modos *decodificar*, *introducirErrores* y *corregirCodigo*, debe tener tamaño *longitud* y sigue el mismo formato que el argumento *alpha*. Por último, *numErrores*, que utilizaremos junto con el modo *introducirErrores*, expresando el número de posiciones no nulas que deseamos en el vector de errores. Es un entero no negativo y debe ser menor o igual que *longitud*. Veamos ahora cada uno de los métodos que componen la clase *GRS*:

- *Constructor*: Como parámetros recibe los siete primeros argumentos que el usario pasa por línea de comandos, junto a dos bits que sirven para indicarnos si la lista de componentes de α y de v han sido pasadas como elementos del cuerpo en cuestión (se usará para definir el código del criptosistema de McEliece) o hay que procesarlas (como cuando se envían por línea de comandos). La función auxiliar *procesarVectores* nos servirá tanto en el constructor como en el resto de métodos de la clase para expresar, en los cuerpos no primos, las componentes del vector pasado como argumento en términos de una base del cuerpo, en caso de que no lo estuvieran ya. El constructor inicializa todas las variables de instancia asociadas con cada uno de los parámetros, realizando las validaciones pertinentes, y finalmente genera la matriz generadora del código en cuestión, siguiendo la estructura de (3.3).

- *Codificar*: Como parámetros recibe el mensaje que deseamos codificar y una bandera que tiene un significado similar al comentado para el caso de *Constructor*, pero ahora para *mensaje*. Tras llevar a cabo el procesamiento y validaciones necesarias, la función devuelve el producto de *mensaje* y la matriz generadora asociada a la instancia, que es la forma de codificar que hemos estudiado para cualquier código lineal.

- *Decodificar*: Sigue una estructura similar a la de *Codificar*, cambiando *mensaje* por *palabraCodigo*. Lo que sí varía es lo que devuelve, ya que en el decodificado partimos

del sistema $m \cdot G = c$, con G y c conocidas. Como G no es invertible (es de orden $k \times n$, luego no es cuadrada), tenemos que calcularle una "pseudo-inversa"por la derecha, resolviendo el sistema $G \cdot X = I_k$, de manera que $m = c \cdot X$.

- *IntroducirErrores*: A diferencia de los métodos anteriores, que eran de instancia, este es estático (necesitaremos invocarlo durante el cifrado del criptosistema de McEliece, donde no podemos manejar una instancia de la clase ya que estaríamos exponiendo la clave privada). A partir del número de errores pasado como argumento, generaremos de manera aleatoria un vector con dicho peso, y lo sumaremos a la palabra código en cuestión.

- *CorregirCodigo*: Implementación del algoritmo para localizar los errores presentes en una palabra código y devolverla corregida. En primer lugar, como sabemos que la capacidad correctora de un código RS generalizado es de hasta $\frac{n-k}{2}$ errores, si el numerador es nulo no tiene sentido ejecutar el algoritmo. De acuerdo con el teorema 3.4.7, necesitamos declarar z^r y $S(z)$. Para construir $S(z)$ usaremos el teorema 3.4.3 junto con el teorema 3.4.2, para obtener los u_is. Una vez construido $S(z)$ podemos ejecutar el algoritmo descrito en el enunciado del teorema 3.4.7, obteniendo los polinomios localizador y evaluador de errores, $\sigma(z)$ y $\omega(z)$ respectivamente. Con $\sigma(z)$ y $\omega(z)$ ya calculados, obtendremos en primer lugar las raíces de $\sigma(z)$ (solo su valor, las multiplicidades no nos interesan), calcularemos sus inversas y buscaremos la intersección de este último conjunto de inversas con las componentes de α, para así determinar el conjunto B de posiciones de los errores en la palabra código, de acuerdo con el punto 2 de la proposición 3.4.4. Realmente, lo que hemos obtenido es el conjunto B_0 y no B, ya que no podemos descartar la presencia de componentes nulas en α. Para contemplar este caso, ya que disponemos del vector α, buscamos si hubiera algún índice con valor nulo y, si además $gr\ \sigma(z) = gr\ \omega(z)$, por el punto 4 de la proposición 3.4.4, sabremos que dicho índice se corresponde con una posición no nula del vector de error, que calcularemos basándonos en el punto 5 de dicha proposición.

Con el fin de validar el correcto funcionamiento de las funciones anteriores, vamos a realizar una prueba de integración que simula el flujo de trabajo habitual de un codificador-decodificador (codificado de mensaje, introducción de errores, corrección de dichos errores y decodificado al mensaje original), para diferentes combinaciones de parámetros. Se define en el archivo *validacionEncDecGRS.py*. Requiere solamente de un parámetro para su ejecución, *numIteraciones*, que indica cuántas iteraciones se desean realizar para cada combinación de parámetros definida en el código, es de tipo entero y hay que introducirla obligatoriamente. Cada combinación consta de longitud y dimensión del código, junto con el orden del cuerpo asociado, además de polinomio irreducible y subcuerpo primo en caso de que fueran necesarios. Para cada una, vamos a medir el número de éxitos y el tiempo medio de ejecución para las iteraciones solicitadas, entendiendo un éxito como aquellos casos donde el mensaje generado aleatoriamente y el devuelto tras decodificar coincidan. En la tabla 3.1 se recogen los resultados de la ejecución de *validacionEncDecGRS.py* con 30 iteraciones. Dicha ejecución se puede visualizar en las figuras 3.1 y 3.2. El cuerpo de orden 81 lo hemos construido como $\mathbb{F}_{81} \cong \frac{\mathbb{F}_3[x]}{2x^4+x^3+1}$.

Longitud	Orden	Dimensión	Iteraciones	Nº Éxitos	% Éxitos	Tiempo Medio (Seg)
20	81	8	30	30	100	0.034484
60	127	28	30	30	100	0.009323
80	149	30	30	30	100	0.014300
96	193	45	30	30	100	0.021299
128	251	64	30	30	100	0.037564
160	257	80	30	30	100	0.064031
200	281	100	30	30	100	0.115764
240	293	120	30	30	100	0.177845

Tabla 3.1: Resultados simulación codificador-decodificador Reed-Solomon generalizados para 30 iteraciones con diferentes parámetros

```
higinio@higinio-PS42-Modern-8RC:~/Plantilla-TFG-master/code$ python3 validacionEncDecGRS.py --numIteraciones 30
Longitud del código: 20
Orden del cuerpo: 81
Polinomio base: 2*x^4 + x^3 + 1
Oden Subcuerpo primo: 3
Dimensión del código: 8
Número de iteraciones totales: 30
Número de éxitos: 30
Porcentaje de aciertos: 100.0
Tiempo medio de ejecución: 0.034484 segundos

Longitud del código: 60
Orden del cuerpo: 127
Dimensión del código: 28
Número de iteraciones totales: 30
Número de éxitos: 30
Porcentaje de aciertos: 100.0
Tiempo medio de ejecución: 0.009323 segundos

Longitud del código: 80
Orden del cuerpo: 149
Dimensión del código: 30
Número de iteraciones totales: 30
Número de éxitos: 30
Porcentaje de aciertos: 100.0
Tiempo medio de ejecución: 0.014300 segundos

Longitud del código: 96
Orden del cuerpo: 193
Dimensión del código: 45
Número de iteraciones totales: 30
Número de éxitos: 30
Porcentaje de aciertos: 100.0
Tiempo medio de ejecución: 0.021299 segundos

Longitud del código: 128
Orden del cuerpo: 251
Dimensión del código: 64
Número de iteraciones totales: 30
Número de éxitos: 30
Porcentaje de aciertos: 100.0
Tiempo medio de ejecución: 0.037564 segundos

Longitud del código: 160
Orden del cuerpo: 257
Dimensión del código: 80
Número de iteraciones totales: 30
Número de éxitos: 30
Porcentaje de aciertos: 100.0
Tiempo medio de ejecución: 0.064031 segundos
```

Figura 3.1: Resultados ejecución simulación codificador-decodificador Reed-Solomon generalizados para 30 iteraciones con diferentes parámetros

Figura 3.2: Continuación ejecución figura 3.1

4 Criptografía post-cuántica

En este cuarto capítulo del trabajo, introduciremos qué es la criptografía, describiendo los dos principales tipos que hay, a los que brevemente daremos formalismo. Finalmente hablaremos de la criptografía post-cuántica, que es donde podemos ubicar al Criptosistema de McEliece. El capítulo se basa fundamentalmente en el contenido de [Vau23].

4.1. ¿Qué es la criptografía?

Dejamos de momento de lado el estudio de la teoría de códigos para adentrarnos en el estudio de la criptografía, el otro gran pilar sobre el que reside la base teórica del trabajo.

Mientras que la teoría de códigos se centra en el problema de asegurar la fiabilidad en la transmisión de la información, añadiendo redundancia para proteger contra el ruido aleatorio, la criptografía se centra en la protección frente a potenciales adversarios que pueden surgir durante dicha transmisión.
A menudo se usa el término criptología como una noción más amplia que criptografía. Este concepto engloba a la propia criptografía, más enfocada al diseño, y al criptoanálisis, más enfocado al análisis de la seguridad de los sistemas. Frecuentemente se piensa en el criptoanálisis como la acción de romper sistemas. Sin embargo, este es solo uno de los aspectos que cubre. Analizando, podemos evidenciar falta de seguridad rompiendo el sistema, pero también podemos demostrar su seguridad.
La criptografía moderna no se centra solamente en la confidencialidad. Muchos otros problemas son considerados: protección de la integridad de la información, autenticidad, no repudio... Estos problemas aparecen de forma natural en muchas aplicaciones del día a día como tarjetas de banco, comercio electrónico, telefonía móvil, pasaportes biométricos, comunicaciones móviles, trazabilidad en cadenas de suministros, televisión de pago, cierres de vehículos, tarifas de transporte público, voto electrónico...
Sin embargo, los tres problemas fundamentales que se encuentran en la comunicación entre un emisor y un receptor son:

- confidencialidad (solo el receptor debe recibir el mensaje)
- autenticidad (solo el emisor debe ser capaz de enviar el mensaje)
- integridad (el mensaje enviado debe coincidir con el recibido)

Encontramos dos grandes grupos dentro de las primitivas criptográficas. En primer lugar, tenemos las primitivas que pertenecen a las técnicas de criptografía simétrica. Entre ellas encontramos al encriptado simétrico, en el cual se encriptan y desencriptan los mensajes con la misma clave, y el código de autenticación de mensaje, donde se calcula una etiqueta para el mensaje utilizando una clave y se verifica dicha etiqueta usando la misma clave. Este último se usa para autenticar mensajes. En ambos casos, la clave utilizada debe mantenerse secreta. A menudo incluimos también las funciones hash en esta categoría de primitivas. Se

trata de una función determinista que calcula una cadena de bits de longitud fija para cualquier mensaje. La salida es una especie de "huella" del mensaje, es decir, única para cada entrada.
En segundo lugar, tenemos la técnica de criptografía de clave pública o asimétrica. Engloba a criptosistemas en los cuales el encriptado y desencriptado son hechos con claves diferentes, siendo la clave de desencriptado secreta al igual que en el caso simétrico. Lo que es nuevo es que la clave de encriptado puede ser hecha pública sin comprometer la privacidad. Otro ejemplo de este tipo de criptografía son las firmas digitales, en las cuales una clave secreta es usada para firmar un mensaje y la clave pública es usada para verificar que la firma es válida. Consideramos también dentro de este grupo los protocolos de acuerdo de clave, los cuales permiten a dos participantes establecer una clave secreta común sobre un canal de comunicación pública.

Ejemplo 4.1.1. En el famoso estándar TLS, que es usado principalmente para navegación segura en Internet, cuando un cliente (navegador) quiere conectarse con un servidor seguro, el servidor envía en primer lugar su certificado, el cual es un documento firmado que incluye la asociación entre la dirección del servidor (la URL) y una clave pública. El navegador sabe cómo verificar el certificado, ya que la lista de claves públicas de las autoridades certificadoras ya está construida. De esta forma, el navegador puede confiar en que una URL está asociada a una determinada clave pública. El navegador puede seleccionar entonces una clave simétrica y encriptarla usando la clave pública del servidor. El servidor es capaz de desencriptarla y recuperar la clave simétrica. Con esta clave simétrica, el cliente y el servidor pueden comunicarse de manera segura usando criptografía simétrica.

El ejemplo anterior refleja cómo en la práctica los dos tipos de criptografía se utilizan de manera conjunta: la criptografía de clave pública es usada para comenzar la comunicación segura y dicha comunicación es realizada usando criptografía simétrica.
A continuación procedemos a darle formalismo a los conceptos ya introducidos de criptografía simétrica y asimétrica.

4.2. Criptografía simétrica

En primer lugar, definimos el cifrado por bloques. Por simplicidad, vamos a trabajar solamente con cadenas de bits, pues se trata del caso más común.

Definición 4.2.1 (Cifrado por Bloques)**.** Un cifrado por bloques es una 4-tupla $(\{0,1\}^k, \{0,1\}^n, Enc, Dec)$, donde $\{0,1\}^k$ es el dominio de la clave, $\{0,1\}^n$ es el dominio de los bloques y Enc, Dec son dos algoritmos deterministas eficientes que verifican

$$\forall K \in \{0,1\}^k \; \forall X \in \{0,1\}^n \; Dec(K, Enc(K, X)) = X.$$

Denotaremos $C_K(\cdot) = Enc(K, .)$ y $C_K^{-1}(\cdot) = Dec(K, .)$.

Por lo tanto, k es la longitud de la clave y n es la longitud del bloque de mensaje. También cabe subrayar que el encriptado y el desencriptado deben ser deterministas, es decir, dada una entrada, siempre deben producir la misma salida para dicha entrada.
Desde un punto de vista más algebraico, de la definición anterior podemos decir que C_K y C_K^{-1} son permutaciones sobre el espacio $\{0,1\}^n$, siendo una la inversa de la otra, $\forall K \in \{0,1\}^k$.

Sin embargo, si nos planteamos cómo podríamos aplicar el cifrado por bloques en un contexto real, se nos presenta el problema de que los textos que recibiría nuestro programa tendrían longitudes diferentes. Por lo tanto, necesitamos un cifrado que cubra diferentes longitudes de entrada. Estos sistemas de cifrado suelen preservar la longitud, de manera que el texto cifrado y el texto plano tienen siempre la misma. Podríamos definir directamente el cifrado sobre el conjunto de todas las cadenas de bits, pero en ocasiones, algunas longitudes de mensaje no están permitidas por la aplicación en cuestión. En consecuencia, debemos definir nuestro sistema de cifrado sobre un subconjunto D del conjunto de cadenas de bits $\{0,1\}^*$.

Definición 4.2.2. Un sistema de cifrado simétrico (de longitud variable y preservando la longitud) es definido como una 4-tupla $(\{0,1\}^k, D, Enc, Dec)$, donde $\{0,1\}^k$ es el dominio de la clave, $D \subseteq \{0,1\}^*$ es el dominio de los textos planos y Enc, Dec son dos algoritmos deterministas eficientes que verifican

$$\forall K \in \{0,1\}^k \; \forall X \in D \begin{cases} Dec(K, Enc(K, X)) = X \\ \text{|Enc(K,X)|} = \text{|X|} \end{cases}.$$

Denotaremos $C_K(\cdot) = Enc(K, .)$ y $C_K^{-1}(\cdot) = Dec(K, .)$.

A menudo, muchos sistemas criptográficos simétricos tienen una entrada adicional que es usada como un número único, a la que se suele denominar valor de seguridad. Por lo tanto, debemos extender nuestra definición.

Definición 4.2.3. Un sistema de cifrado simétrico (basado en valor de seguridad, de longitud variable y preservando la longitud) es definido como una 5-tupla $(\{0,1\}^k, D, \mathcal{N}, Enc, Dec)$, donde $\{0,1\}^k$ es el dominio de la clave, $D \subseteq \{0,1\}^*$ es el dominio de los textos planos, $\mathcal{N}$ es el dominio de los valores de seguridad y Enc, Dec son dos algoritmos deterministas eficientes que verifican

$$\forall K \in \{0,1\}^k \; \forall X \in D \; \forall N \in \mathcal{N} \begin{cases} Dec(K, N, Enc(K, N, X)) = X \\ \text{|Enc(K,N,X)|} = \text{|X|} \end{cases}$$

Suponemos que cada N es usado como mucho una vez en nuestro criptosistema. Podríamos usar un valor de seguridad aleatorio, con un dominio de muestreo lo suficientemente grande para evitar duplicados, o un contador. También se puede enviar el valor de seguridad en plano junto con el texto cifrado, o tener al emisor y al receptor sincronizados en dicho valor por otros medios.

4.3. Criptografía asimétrica

En esta breve sección vamos a ver la definición de criptosistema de clave pública, del cual el criptosistema de McEliece es un ejemplo.

Definición 4.3.1 (Criptosistema de Clave Pública)**.** Un criptosistema de clave pública es una 4-tupla (Gen, M, Enc, Dec), donde $M \subseteq \{0,1\}^*$ es el dominio de los textos planos y Gen, Enc, Dec son tres algoritmos eficientes. El algoritmo Dec es determinista y su imagen es $\{M \cup \perp\}$, donde $\perp$ denota que se ha producido un error. Se verifica

$$\forall X \in M \; Pr[Dec(sk, Enc(pk, X)) = X] = 1,$$

donde (pk, sk) es generado al ejecutar *Gen*. La probabilidad es sobre la aleatoriedad introducida en los algoritmos *Gen* y *Enc*.

pk será la clave pública del criptosistema, mientras que *sk* será la privada. A diferencia de un criptosistema de clave simétrica, donde todos los algoritmos debían ser deterministas, aquí tan solo le imponemos esa condición al algoritmo de descifrado (lo que tiene sentido pues, en caso contrario, un texto cifrado podría dar lugar a varios textos planos), mientras que los de generación de claves y de cifrado pueden ser probabilísticos, de manera que dada una entrada, existe una distribución de probabilidad sobre las posibles salidas.

4.4. Criptografía post-cuántica

La mayoría de criptosistemas de clave pública clásicos están basados en los problemas de la factorización (dado un número entero, descomponerlo en sus factores primos, como es el caso de RSA) y del logaritmo discreto (dado un grupo G de orden n, un elemento g generando a G y un elemento y de g, encontrar un entero x tal que $g^x = y$, como es el caso de ElGamal). Ambos problemas, teóricamente, pueden ser resueltos de forma eficiente por un ordenador cuántico suficientemente potente. Cuando estos dispositivos estén disponibles (aunque es imposible predecir cuándo y si será posible), se presupone que gran parte de los sistemas actuales serán inseguros. Por ello, necesitamos esquemas alternativos basados en problemas matemáticos diferentes y más desafiantes para la computación cuántica.
Hasta ahora, las direcciones más prometedoras se basan en problemas de la teoría de códigos (como el criptosistema de McEliece que vamos a estudiar) o de la teoría de retículos. Un retículo del espacio vectorial R^n se define como el conjunto de vectores que se obtienen mediante combinaciones lineales, con coeficientes enteros, de una base de R^n. Es decir, dada una base $\{v_1, ..., v_n\}$, el retículo Λ que generan es

$$\Lambda = \{a_1 v_1 + a_2 v_2 + ... + a_n v_n | a_1, a_2, ..., a_n \in \mathbb{Z}\}$$

En esta estructura, dada la base $\{v_1, ..., v_n\}$, es difícil encontrar el vector más pequeño (en términos de una norma) de R^n que se encuentre en Λ. Dado además un vector $b \in R^n$, es también difícil encontrar el vector $v \in \Lambda$ más cercano (en términos de una distancia sobre R^n). La NP-Completitud de estos problemas hace que muchos criptosistemas estén basados en retículos.

5 Criptosistema de McEliece

Pasamos ahora al capítulo principal del trabajo. Empezaremos revisando la propuesta original del criptosistema de McEliece, basada en códigos Goppa. Debido a ciertas limitaciones que presenta, plantearemos como alternativa el uso de códigos Reed-Solomon generalizados, que ya vimos anteriormente, y de la que además propondremos una posible implementación. Como referencias, usaremos [Val] y [D.Eo7].

5.1. Criptosistema de McEliece basado en códigos Goppa

Comencemos viendo la definición de código Goppa.

Definición 5.1.1. Sea $g(x) \in \mathbb{F}_q[x]$, donde $q = p^m$, con p primo y $m \in \mathbb{N}^*$, de la forma $g(x) = \sum_{i=0}^{t} g_i x^i$, con $t \in \mathbb{N}^*$. Sea también $L = \{\alpha_1, ..., \alpha_n\} \subseteq F_q$, de manera que $g(\alpha_i) \neq 0, \forall \alpha_i \in L$. Dado un vector $c = (c_1, ..., c_n) \in \mathbb{F}_q^n$, definimos la función $R_c(x) = \sum_{i=1}^{n} \frac{c_i}{x-\alpha_i}, \forall x \in \mathbb{F}_q$, siendo $\frac{1}{x-\alpha_i}$ el inverso de $x - \alpha_i$ en el anillo cociente $\frac{\mathbb{F}_q[x]}{g(x)}$. Las palabras código que conformen el código Goppa $\Gamma(L, g(x))$ serán aquellos vectores c que verifiquen $R_c(x) \equiv 0 \ (mod g(x))$, es decir, cuando $R_c(x)$ sea múltiplo de $g(x)$.

Mientras que la longitud del código $\Gamma(L, g(x))$ será simplemente el cardinal de L, para la dimensión no podemos dar un valor exacto de manera general, aunque el siguiente teorema sí que nos da una cota inferior.

Teorema 5.1.1. *La dimensión k de un código Goppa $\Gamma(L, g(x))$ es mayor o igual que $n - mt$, es decir, $k \geq n - mt$.*

Para nuestro propósito, vamos a imponer que $p = 2$ y que el polinomio $g(x)$ sea irreducible sobre $F_q[x]$. El siguiente teorema va a justificar esta elección.

Teorema 5.1.2. *Sea $\Gamma(L, g(x))$ un código Goppa definido sobre un cuerpo binario y donde $g(x)$ es irreducible. Si denotamos por d a su peso mínimo, tenemos que $d \geq 2t + 1$, con t el grado de $g(x)$.*

Demostración. Sea $c = (c_1, ..., c_n) \in \Gamma(L, g(x))$ una palabra código y $M_c = \{i : c_i = 1\}$. Definimos el siguiente polinomio

$$\sigma_c(x) = \prod_{j \in M_c} (x - \alpha_j) \in \mathbb{F}_q[x],$$

cuya derivada es

$$\sigma_c'(x) = \sum_{i \in M_c} \prod_{j \in M_c \setminus \{i\}} (x - \alpha_j).$$

Se da la igualdad

$$\sigma_c(x) R_c(x) \equiv \sigma_c'(x) \text{ (mod g(x))}.$$

Como por definición $g(\alpha_i) \neq 0$, $\forall 1 \leq i \leq n$, tenemos que $mcd(\sigma_c(x), g(x)) = 1$, ya que ambos polinomios no tienen ninguna raíz común y $g(x)$ es irreducible. Por lo tanto, $\sigma_c(x)$ es invertible módulo $g(x)$ y podemos reescribir la igualdad anterior

$$\frac{\sigma_c'(x)}{\sigma_c(x)} \equiv R_c(x) \text{ (mod g(x))}.$$

Deducimos la siguiente condición suficiente y necesaria para que un vector de $\mathbb{F}_q^n$ sea una palabra código de $\Gamma(L, g(x))$:

$$\forall c \in \mathbb{F}_q^n : c \in \Gamma(L, g(x)) \iff \sigma_c'(x) \equiv 0 \text{ (mod g(x))}.$$

La aplicación $F_q \to F_q, x \to x^2$ es el automorfismo de Frobenius para F_q. En particular, es inyectiva y sobreyectiva, de manera que $\forall y \in F_q$, existe una única raíz cuadrada en F_q. Podemos extender esta aplicación de la siguiente manera, teniendo en cuenta que al trabajar sobre un cuerpo binario se da la igualdad $(a+b)^2 = a^2 + b^2, \forall a, b \in \mathbb{F}_q$

$$F_q[x] \to F_q[x], f(x) = \sum_{i=0}^{n} f_i x^i \to (f(x))^2 = \sum_{i=0}^{n} f_i^2 x^{2i}.$$

Esta aplicación es un homomorfismo de anillos inyectivo (al existir una única raíz cuadrada para cada elemento), pero no sobreyectivo (ya que existen polinomios de grado impar en $\mathbb{F}_q[x]$). Su imagen es $F_q[x^2]$, el conjunto de cuadrados perfectos del anillo $\mathbb{F}_q[x]$, es decir, $\forall p(x) \in \mathbb{F}_q[x^2]\ \exists q(x) \in \mathbb{F}_q[x] : p(x) = q(x)^2$.
Si expresamos el polinomio $\sigma_c(x)$ como $\sigma_c(x) = \sum_{i=0}^{n} \sigma_i x^i$, tenemos que $\sigma_c'(x) = \sum_{i=1}^{n} i\sigma_i x^{i-1}$ es un cuadrado perfecto, ya que todos los términos impares desaparecen al verificarse $i\sigma_i x^{i-1} = 0$ para todo i par, al estar trabajando sobre un cuerpo binario. Sea $h(x) \in \mathbb{F}_q[x]$ de manera que $\sigma_c'(x) = h(x)^2$. Sabemos ya que $g(x) | \sigma_c'(x) = h(x)^2$, y como $g(x)$ es irreducible y $\mathbb{F}_q[x]$ es un DFU, se verifica que $g(x) | h(x)$. Pero entonces existe $q(x) \in \mathbb{F}_q[x]$ tal que $h(x) = q(x)g(x)$, luego $\sigma_c'(x) = q(x)^2 g(x)^2$, obteniendo

$$\forall c \in \mathbb{F}_q^n : c \in \Gamma(L, g(x)) \iff \sigma_c'(x) \equiv 0\ (mod g^2(x)).$$

Concluimos que, $\forall c \in \Gamma(L, g(x)) \setminus \{0\}$, tenemos

$$wt(c) = \text{gr } \sigma_c(x) \geq \text{gr } \sigma_c'(x) + 1 \geq \text{gr } g^2(x) + 1 \geq 2t + 1.$$

□

Sabemos que si un código lineal tiene peso mínimo d, entonces podrá corregir hasta $[\frac{d-1}{2}]$ errores. Utilizando el teorema 5.1.2, tenemos que un código Goppa binario irreducible podrá corregir hasta t errores. A pesar de disponer de una cota, necesitamos también un algoritmo para corregir dichos errores. Para ello, usaremos el algoritmo de Patterson.

Sea y una palabra código recibida con $r \leq t$ errores,
$y = (y_1, ..., y_n) = (c_1, ..., c_n) + (e_1, ..., e_n) = c + e$, con $wt(e) = r$. Para obtener c a partir de y, debemos encontrar las posiciones no nulas del vector e (el valor no es necesario ya que estamos en el caso binario), es decir, el conjunto $E = \{i : e_i \neq 0\}$. Definimos el polinomio localizador de errores $\sigma(x)$ como $\sigma(x) = \prod_{i \in E}(x - \alpha_i)$. Al estar trabajando en binario, nos bastará con un solo polinomio en el algoritmo de corrección, a diferencia, por ejemplo, del

caso de los códigos Reed-Solomon generalizados, donde usábamos dos. El algoritmo de Patterson es el siguiente:

1. Calculamos el polinomio síndrome $R_y(x) = \sum_{i=1}^{n} \frac{y_i}{x-\alpha_i} (mod\ g(x))$.
2. Calculamos el polinomio localizador de errores siguiendo los siguientes pasos:
   a) Buscamos el inverso del polinomio síndrome en $\frac{\mathbb{F}_q[x]}{g(x)}$, es decir, un $h(x)$ tal que $R_y(x)h(x) \equiv 1(modg(x))$.
   b) Buscamos $d(x)$ que verifique $d^2(x) \equiv h(x) + x(mod\ g(x))$.
   c) Buscamos $a(x)$ y $b(x)$ que cumplan $d(x)b(x) \equiv a(x)(mod\ g(x))$. Debe verificarse gr $a(x) \leq \frac{t}{2}$, gr $b(x) \leq \frac{t-1}{2}$, ya que gr $\sigma(x) \leq t$.
   d) Finalmente, $\sigma(x) = a^2(x) + xb^2(x)$.
3. Una vez calculado $\sigma(x)$, podemos obtener el conjunto E, que es igual a $E = \{i : \sigma(\alpha_i) = 0\}$.
4. El vector de error e estará definido como $e_i = 1$, si $i \in E$, y $e_i = 0$ en caso contrario.
5. La palabra código original será (en caso de que se hayan cometido t o menos errores) $c = y - e$.

Veamos que este algoritmo efectivamente nos conduce a la obtención del polinomio localizador de errores. Como c es una palabra código, tenemos que $R_c(x) \equiv 0\ (modg(x))$ y por lo tanto

$$R_y(x) \equiv R_e(x) \text{ (mod g(x))}.$$

Mediante un razonamiento similar al de la demostración del teorema 5.1.2, obtenemos que

$$\sigma(x)R_e(x) \equiv \sigma'(x) \text{ (mod g(x))}. \tag{5.1}$$

Podemos ahora dividir el polinomio $\sigma(x)$ en cuadrados y no cuadrados, ya que al estar trabajando sobre un cuerpo binario, sabemos que todo elemento puede expresarse como cuadrado de otro:

$$\sigma(x) = a^2(x) + xb^2(x), \sigma'(x) = 2a(x)a'(x) + b^2(x) + 2xb(x)b'(x) = b^2(x).$$

La ecuación (5.1) puede ser reescrita de la siguiente manera:

$$\begin{aligned}(a^2(x) + xb^2(x))R_e(x) &\equiv b^2(x) \text{ (mod g(x))},\\ b^2(x) + xb^2(x)R_e(x) &\equiv a^2(x)R_e(x) \text{ (mod g(x))},\\ b^2(x)(xR_e(x) + 1) &\equiv a^2(x)R_e(x) \text{ (mod g(x))}.\end{aligned}$$

Podemos suponer que e no es una palabra código, por lo que $R_e(x) \not\equiv 0\ (modg(x))$. Por ser $g(x)$ irreducible, $\frac{\mathbb{F}_q[x]}{g(x)}$ es un cuerpo, luego existe el inverso de $R_e(x)$ módulo $g(x)$, $h(x) = R_e(x)^{-1}$. Si multiplicamos por $h(x)$ en ambos lados de la última equivalencia:

$$b^2(x)(x + h(x)) \equiv a^2(x) \text{ (mod g(x))}. \tag{5.2}$$

Sea ahora $d(x) \in \mathbb{F}_q[x]$ la única raíz cuadrada del polinomio $h(x) + x$, es decir, $d(x)d(x) \equiv h(x) + x \ (modg(x))$. Tomando ahora raíces cuadradas de la ecuación (5.2) obtenemos

$$d(x)b(x) \equiv a(x) \text{ mod(g(x))}.$$

Dados $d(x)$ y $g(x)$, tenemos que calcular $a(x)$ y $b(x)$. Vemos que esta ecuación se asemeja bastante a la ecuación clave que teníamos en los códigos GRS ($\sigma(z)S(z) \equiv \omega(z) \ (modz^r)$), que resolvíamos mediante el uso del algoritmo de Euclides. Esta ecuación tendrá una resolución similar. Finalmente, calculamos el polinomio localizador de errores $\sigma(x) = a^2(x) + xb^2(x)$.

No es sencillo dar la forma explícita de una matriz generadora para un código Goppa. Por lo tanto, lo que haremos es partir de una matriz de paridad y, usando que todas las palabras código deben estar en el núcleo de dicha matriz, construiremos una base de dicho núcleo, cuyos elementos serán las filas de una matriz generadora para el mismo código. La siguiente proposición nos da una posible construcción de matriz de paridad.

Proposición 5.1.3. *Sea $H = XYZ$ con*

$$X = \begin{pmatrix} g_t & 0 & 0 & \dots & 0 \\ g_{t-1} & g_t & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ g_1 & g_2 & g_3 & \dots & g_t \end{pmatrix}, Y = \begin{pmatrix} 1 & 1 & \dots & 1 \\ \alpha_1 & \alpha_2 & \dots & \alpha_n \\ \vdots & \vdots & \ddots & \vdots \\ \alpha_1^{t-1} & \alpha_2^{t-1} & \dots & \alpha_n^{t-1} \end{pmatrix}, Z = \begin{pmatrix} \frac{1}{g(\alpha_1)} & 0 & \dots & 0 \\ 0 & \frac{1}{g(\alpha_2)} & \ddots & \vdots \\ \vdots & \vdots & \ddots & 0 \\ 0 & \dots & 0 & \frac{1}{g(\alpha_n)} \end{pmatrix}.$$

Entonces H es una matriz de paridad para el código Goppa $\Gamma(L, g(x))$.

Demostración. En primer lugar, cabe mencionar que $R_c(x)$ se puede ver también como $R_c(x) \equiv -\sum_{i=1}^{n} \frac{c_i}{g(\alpha_i)} \frac{g(x)-g(\alpha_i)}{x-\alpha_i} \ (mod(g(x))$. Se verifica, aplicando la fórmula del cociente de diferencia de potencias y cambiando de variable:

$$\frac{g(x) - g(\alpha_i)}{x - \alpha_i} = \sum_{j=0}^{t} g_j \frac{x^j - \alpha_i^j}{x - \alpha_i} = \sum_{s=0}^{t-1} x^s \sum_{j=s+1}^{t} g_j \alpha_i^{j-1-s} \text{ para todo } 1 \leq i \leq n.$$

Vemos que una palabra código $c = (c_1, ..., c_n)$ pertenecerá a $\Gamma(L, g(x))$ si y solo si, $\forall s = 0, ..., t-1$

$$\sum_{i=1}^{n} (\frac{1}{g(\alpha_i)} \sum_{j=s+1}^{t} g_j \alpha_i^{j-1-s}) c_i = 0.$$

Esta caracterización nos sugiere la estructura de una posible matriz de paridad

$$H = \begin{pmatrix} g_t g(\alpha_1)^{-1} & \dots & g_t g(\alpha_n)^{-1} \\ (g_{t-1} + g_t\alpha_1)g(\alpha_1)^{-1} & \dots & (g_{t-1} + g_t\alpha_n)g(\alpha_n)^{-1} \\ \vdots & \ddots & \vdots \\ (\sum_{j=1}^{t} g_j \alpha_1^{j-1})g(\alpha_1)^{-1} & \dots & (\sum_{j=1}^{t} g_j \alpha_n^{j-1})g(\alpha_n)^{-1} \end{pmatrix} = XYZ,$$

donde

$$X = \begin{pmatrix} g_t & 0 & 0 & \dots & 0 \\ g_{t-1} & g_t & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ g_1 & g_2 & g_3 & \cdots & g_t \end{pmatrix}, Y = \begin{pmatrix} 1 & 1 & \dots & 1 \\ \alpha_1 & \alpha_2 & \dots & \alpha_n \\ \vdots & \vdots & \ddots & \vdots \\ \alpha_1^{t-1} & \alpha_2^{t-1} & \dots & \alpha_n^{t-1} \end{pmatrix}, Z = \begin{pmatrix} \frac{1}{g(\alpha_1)} & 0 & \dots & 0 \\ 0 & \frac{1}{g(\alpha_2)} & \ddots & \vdots \\ \vdots & \vdots & \ddots & 0 \\ 0 & \dots & 0 & \frac{1}{g(\alpha_n)} \end{pmatrix}.$$

□

Veamos un ejemplo para ilustrar cómo funcionaría el proceso de codificación-decodificación con un código Goppa.

Ejemplo 5.1.1. Vamos a trabajar sobre $\mathbb{F}_4$. Utilizaremos su representación como el cociente $\mathbb{F}_4 \cong \frac{\mathbb{F}_2[x]}{x^2+x+1}$, de manera que sus elementos son $\mathbb{F}_4 = \{0, 1, \alpha, \alpha+1\}$, siendo α una raíz de x^2+x+1 en $\mathbb{F}_4$. Como polinomio irreducible, vamos a escoger $g(x) = x^2 + x + \alpha \in \mathbb{F}_4[x]$. En efecto, es irreducible ya que al ser de grado 2 debería tener un factor lineal:

$$\begin{aligned} g(0) &= 0^2 + 0 + \alpha = \alpha \neq 0, \\ g(1) &= 1^2 + 1 + \alpha = 1 + 1 + \alpha = 0 + \alpha = \alpha \neq 0, \\ g(\alpha) &= \alpha^2 + \alpha + \alpha = \alpha^2 + 2\alpha = \alpha^2 = \alpha + 1 \neq 0, \\ g(\alpha+1) &= (\alpha+1)^2 + (\alpha+1) + \alpha \\ &= \alpha + (\alpha+1) + \alpha = \alpha + \alpha + 1 + \alpha = (0 + 1 + \alpha) = 1 + \alpha \neq 0. \end{aligned}$$

Por lo tanto, $t = 2$. Como subconjunto L vamos a escoger $L = \{1, \alpha, \alpha+1\}$, que verifica $g(1) \neq 0, g(\alpha) \neq 0$ y $g(\alpha+1) \neq 0$. Ahora utilicemos la proposición 5.1.3 para obtener una matriz de paridad para el código Goppa definido por L y g.

$$X = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}, Y = \begin{pmatrix} 1 & 1 & 1 \\ 1 & \alpha & \alpha+1 \end{pmatrix}, Z = \begin{pmatrix} \frac{1}{\alpha} & 0 & 0 \\ 0 & \frac{1}{\alpha+1} & 0 \\ 0 & 0 & \frac{1}{\alpha+1} \end{pmatrix},$$

$$H = XYZ = \begin{pmatrix} 1 & 1 & 1 \\ 0 & \alpha+1 & \alpha \end{pmatrix} \begin{pmatrix} \frac{1}{\alpha} & 0 & 0 \\ 0 & \frac{1}{\alpha+1} & 0 \\ 0 & 0 & \frac{1}{\alpha+1} \end{pmatrix} = \begin{pmatrix} \frac{1}{\alpha} & \frac{1}{\alpha+1} & \frac{1}{\alpha+1} \\ 0 & 1 & \frac{\alpha}{\alpha+1} \end{pmatrix} = \begin{pmatrix} \alpha+1 & \alpha & \alpha \\ 0 & 1 & \alpha^2 \end{pmatrix} = \begin{pmatrix} \alpha+1 & \alpha & \alpha \\ 0 & 1 & \alpha+1 \end{pmatrix}.$$

Podemos obtener las filas de la matriz generadora G calculando el núcleo de H, que en este caso es $Ker(H) = \{(\lambda, (\alpha+1)\lambda, \lambda) : \lambda \in \mathbb{R}\}$, de manera que una matriz generadora podría ser $G = \begin{pmatrix} 1 & \alpha+1 & 1 \end{pmatrix}$. Supongamos que deseamos codificar el mensaje $m = (\alpha)$. Obtendríamos la palabra código $c = m \cdot G = (\alpha, \alpha^2 + \alpha, \alpha) = (\alpha, 1, \alpha)$. Como $t = 2$, podemos introducir hasta dos errores. Sea por ejemplo $e = (1, 1, 0)$, de forma que $y = c + e = (\alpha, 1, \alpha) + (1, 1, 0) = (\alpha+1, 0, \alpha)$. Pasemos ahora a la corrección de los errores de y, usando el algoritmo de Patterson. Empezamos por el cálculo del polinomio síndrome

$$R_y(x) = \frac{\alpha+1}{x-1} + \frac{\alpha}{x-(\alpha+1)} \pmod{x^2+x+\alpha}. \tag{5.3}$$

Como $x^2 + x + \alpha$ es irreducible en $\mathbb{F}_4[x]$, podemos calcular los inversos que aparecen en la ecuación (5.3) usando el algoritmo extendido de Euclides, de manera que

$$\begin{aligned} R_y(x) &= (\alpha+1)\cdot(\alpha+1)x + \alpha\cdot(\alpha x + \alpha + 1) \pmod{x^2+x+\alpha} \\ &= (\alpha^2+1)x + \alpha^2 x + \alpha^2 + \alpha \pmod{x^2+x+\alpha} \\ &= x + \alpha^2 + \alpha = x + 1 \pmod{x^2+x+\alpha}. \end{aligned}$$

Volvemos a aplicar el algoritmo de Euclides para calcular el inverso de $R_y(x)$,

$$h(x) = (\alpha+1)x \pmod{x^2+x+\alpha}.$$

Una vez obtenido, necesitamos encontrar un polinomio $d(x)$ tal que $d^2(x) = (\alpha+1)x + x = \alpha x \pmod{x^2+x+\alpha}$. Al tratarse de un anillo pequeño, probamos con todos los elementos hasta encontrar uno que lo verifique. En nuestro caso va a ser $d(x) = (\alpha+1)x + \alpha$:

$$\begin{aligned} ((\alpha+1)x+\alpha)^2 = (\alpha+1)^2x^2 + \alpha^2 = (\alpha^2+1)x^2 + \alpha^2 = \alpha x^2 + \alpha^2 = \alpha x^2 + \alpha + 1 = \\ = \alpha(x+\alpha) + \alpha + 1 = \alpha x + \alpha^2 + \alpha + 1 = \alpha x \pmod{x^2+x+\alpha}. \end{aligned}$$

Buscamos $a(x)$ y $b(x)$ tales que $((\alpha+1)x+\alpha)b(x) = a(x) \pmod{x^2+x+\alpha}$. Como debe cumplirse gr $b(x) \leq \frac{t-1}{2} = \frac{2-1}{2} = \frac{1}{2}$, necesariamente gr $b(x) = 0$, luego $b(x) = 1$ y $a(x) = (\alpha+1)x + \alpha$. Sustituyendo ahora en $\sigma(x) = a^2(x) + xb^2(x)$:

$$\sigma(x) = ((\alpha+1)x+\alpha)^2 + x = (\alpha^2+1)x^2 + \alpha^2 + x = \alpha x^2 + x + \alpha + 1.$$

El conjunto E de posiciones de los errores coincide con la posición de las raíces de $\sigma(x)$ en L:

$$\begin{aligned} &\sigma(1) = \alpha + 1 + \alpha + 1 = 0, \\ &\sigma(\alpha) = \alpha^3 + \alpha + \alpha + 1 = \alpha^3 + 1 = \alpha(\alpha+1) + 1 = \alpha^2 + \alpha + 1 = 0. \end{aligned}$$

Es decir, $E = \{1, 2\}$. Por lo tanto, $c = y - e = (\alpha+1, 0, \alpha) - (1, 1, 0) = (\alpha, 1, \alpha)$, y para decodificar c en m hay que tener en cuenta la igualdad $c = m \cdot G = (x) \cdot G$, con x la incógnita, de manera que igualando, por ejemplo, la primera componente de c con la de $m \cdot G$ obtenemos $x = \alpha$, luego $m = (\alpha)$.

Podemos pasar ya a definir el criptosistema de McEliece. Se trata de un criptosistema de clave pública y basado en teoría de códigos. Como parámetros de dicho criptosistema se deberán definir la longitud del código n (que nos indica el tamaño de L) y el grado de $g(x)$, t. La primera aproximación que vamos a tratar está basada en códigos Goppa. Como vimos en la definición de criptosistema de clave pública, para definir uno es necesario proveer tres algoritmos, *Gen*, *Enc* y *Dec*, cuyo pseudocódigo es el siguiente:

Algorithm *Gen*

Seleccionamos de manera aleatoria un polinomio irreducible $g(x) \in F_{2^m}[x]$ de grado t y un subconjunto $L \subseteq F_{2^m}$ de longitud n, que definan un código Goppa. Construimos una matriz generadora G de dimensión $k \times n$ para dicho código.
Seleccionamos de manera aleatoria dos matrices, una de ellas invertible, S, de dimensión $k \times k$, y la otra de permutación, P, de dimensión $n \times n$.
$G' \leftarrow SGP$
$pk \leftarrow (G', t)$
$sk \leftarrow (g(x), L, G, S, P)$
Devolver (sk, pk)

Algorithm $Enc(pk, pt)$

Seleccionamos aleatoriamente un vector binario e de dimensión $1 \times n$, de manera que $wt(e) \leq t$.
$ct \leftarrow ptG' + e$
Devolver ct

Algorithm $Dec(sk, ct)$

$ct' \leftarrow ctP^{-1} = ptG'P^{-1} + eP^{-1} = ptSGPP^{-1} + e' = (ptS)G + e' = pt'G + e'$
Calculamos e' aplicando el algoritmo de Patterson.
$pt'G \leftarrow ct' - e'$
Decodificamos $pt'G$ en pt'.
$pt \leftarrow pt'S^{-1}$
Devolver pt

La condición, durante el cifrado, de que el vector e tenga peso menor o igual a t es para que así sea viable la corrección de errores, de acuerdo a lo visto anteriormente. Además, al multiplicar dicho vector por la inversa de una matriz de permutación (que es también de permutación), ya que el efecto de dicha matriz es simplemente reordenar términos, el peso del vector se mantiene. El texto plano pt debe ser un vector de longitud k, y el texto cifrado ct debe ser un vector de longitud n.
El siguiente ejemplo muestra cómo funcionaría el criptosistema en la práctica.

Ejemplo 5.1.2. Sea $G = \begin{pmatrix} 1 & \alpha+1 & 1 \end{pmatrix}$ la matriz generadora del código Goppa del ejemplo 5.1.1. Como matrices S y P vamos a escoger

$$S = \begin{pmatrix} 1 \end{pmatrix}, P = \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}.$$

Calculamos

$$G' = SGP = \begin{pmatrix} \alpha+1 & 1 & 1 \end{pmatrix}.$$

Si ahora $pt = (\alpha)$ y $e = (1\ 0\ 1)$ ($wt(e) <= t = 2$), el texto cifrado ct sería

$$ct = (\alpha)\,(\alpha+1 \quad 1 \quad 1) + (1 \quad 0 \quad 1) = (\alpha^2+\alpha \quad \alpha \quad \alpha) + (1 \quad 0 \quad 1) = (0 \quad \alpha \quad \alpha+1)\,.$$

Vamos con el descifrado de ct. En primer lugar, multiplicamos a su derecha por P^{-1} (se obtiene como la traspuesta de P):

$$ct' = (0 \quad \alpha \quad \alpha+1)\begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix} = (\alpha+1 \quad 0 \quad \alpha)\,.$$

Como ct' coincide con la palabra que hemos corregido en el ejemplo 5.1.1, sabemos que $e' = (1,1,0)$, luego $pt'G = (\alpha+1,0,\alpha) - (1,1,0) = (\alpha,1,\alpha)$. Decodificamos $pt'G$ en pt' teniendo en cuenta la igualdad $(\alpha,1,\alpha) = pt'G = (x)G$, con x una incógnita, de manera que igualando, por ejemplo, la primera componente de $(\alpha,1,\alpha)$ con la de $pt'G$ obtenemos $x = \alpha$, luego $pt' = (\alpha)$. Finalmente, calculamos pt multiplicando por S^{-1} a la derecha, pero como en este ejemplo es (1), nos queda $pt = pt' = (\alpha)$.

La seguridad de este criptosistema se basa en dos pilares: por un lado, la ya probada NP-Completitud del problema de decodificar un código lineal binario, si no conocemos su estructura; por otro lado, para precisamente no exponer dicha estructura, enmascaramos la matriz generadora del código mediante la multiplicación por una matriz invertible y otra de permutación. Además, es necesario también que el código tenga una longitud y dimensión muy grandes porque, de lo contrario, sería más fácil para un potencial atacante reconocer el código que hay detrás de la matriz generadora pública. Por ejemplo, en su artículo original, McEliece recomendaba el uso de un código Goppa de longitud 1024 y dimensión 524. Esto se traduce a que la clave pública (omitiendo t) ocuparía, como mínimo (suponiendo el caso más simple que es sobre $\mathbb{F}_2$), $524 \cdot 1024 = 536576$ bits $= 65822$ bytes $= 64.38$ KB. Si lo comparamos con una versión actual de RSA, que utiliza claves públicas de entre 2 y 4 KB, podemos ver que ocupa un espacio demasiado grande, generando una transmisión lenta y difícil implementación en sistemas con recursos limitados.

Como posible solución a estos problemas, vamos a proponer el uso de otra familia de códigos que ya hemos estudiado con detenimiento.

5.2. Criptosistema de McEliece basado en códigos Reed-Solomon generalizados

Supongamos que el código Goppa que hemos propuesto anteriormente como ejemplo, de longitud 1024 y dimensión 524, trabajara sobre $\mathbb{F}_{2^{10}}$. Usando el teorema 5.1.1, obtenemos que $524 \geq 1024 - 10t \Rightarrow -500 \geq -10t \Rightarrow 500 \leq 10t \Rightarrow t \geq 50$. Es decir, este código podría corregir al menos 50 errores. Busquemos ahora un código GRS que provea una seguridad similar, $GRS_{n,k}(\alpha, v)$. Recordemos que dicho código puede corregir hasta $\frac{n-k}{2}$ errores. Si imponemos la misma relación entre longitud y dimensión que para el código Goppa ($\frac{524}{1024} = 0.512$), obtenemos la igualdad $k = 0.512n$, que podemos sustituir en la ecuación $\frac{n-k}{2} = 50 \Rightarrow n - k = 100$, de manera que
$(1 - 0.512)n = 100 \Rightarrow 0.488n = 100 \Rightarrow n = 204.92, k = 104.92$, y redondeando ambos a su entero más próximo $n = 205, k = 105$, que cumplen la corrección de hasta 50 errores. En

cuanto al tamaño q del cuerpo base del código GRS, como sabemos que debe verificar $q \geq n = 205$, podemos seguir trabajando sobre cuerpos binarios y elegir $q = 2^8 = 256 \geq 205$. La matriz generadora del código GRS y, por tanto, la clave pública, en caso de su uso en el criptosistema de McEliece, serían mucho más reducidas que con el uso de códigos Goppa y con un nivel de seguridad parecido. Más específicamente, en el caso de uso de códigos Goppa, teniendo en cuenta que un elemento de $\mathbb{F}_{2^{10}}$ ocupa 10 bits (coeficientes de un polinomio de grado menor que 10 sobre $\mathbb{F}_2$), la clave pública G' de orden 524×1024 ocuparía $524 \times 1024 \times 10 = 5365760$ bits $= 670720$ bytes $= 655$ KB. Por otro lado, si utilizamos los Reed-Solomon generalizados, teniendo en cuenta que un elemento de $\mathbb{F}_{2^8}$ ocupa 8 bits (coeficientes de un polinomio de grado menor que 8 sobre $\mathbb{F}_2$), la clave pública G' de orden 105×205 ocuparía $105 \times 205 \times 8 = 172200$ bits $= 21525$ bytes $\approx$ 21KB, es decir, ocupa alrededor del 96 % menos que la clave pública con códigos Goppa.

Los algoritmos *Gen*, *Enc* y *Dec* cambiarían de la siguiente manera:

Algorithm *Gen*

Seleccionamos de manera aleatoria los vectores v y α, con los elementos de v distintos de 0 y los de α distintos entre sí. Construimos una matriz generadora G de dimensión $k \times n$ para dicho código.
Seleccionamos de manera aleatoria dos matrices, una de ellas invertible, S, de dimensión $k \times k$, y la otra de permutación, P, de dimensión $n \times n$.
$G' \leftarrow SGP$
$pk \leftarrow G'$
$sk \leftarrow (\alpha, v, G, S, P)$
Devolver (sk, pk)

Algorithm $Enc(pk, pt)$

Seleccionamos aleatoriamente un vector e de dimensión $1 \times n$, de manera que $wt(e) \leq \frac{n-k}{2}$.
$ct \leftarrow ptG' + e$
Devolver ct

Algorithm $Dec(sk, ct)$

$ct' \leftarrow ctP^{-1} = ptG'P^{-1} + eP^{-1} = ptSGPP^{-1} + e' = (ptS)G + e' = pt'G + e'$
Calculamos e' aplicando el algoritmo de Euclides.
$pt'G \leftarrow ct' - e'$
Decodificamos $pt'G$ en pt'.
$pt \leftarrow pt'S^{-1}$
Devolver pt

Los pseudocódigos anteriores se implementan en el archivo *McElieceGRS.py*.
Necesitaremos tener instalados en nuestro equipo tanto Python como SageMath para poder ejecutarlo. Requiere de 6 parámetros para su ejecución, *longitud*, *ordenCuerpo*, *polinomioBase*, *ordenSubcuerpoPrimo*, *dimension* y *mensaje*, que siguen el mismo formato que sus homónimos descritos para el codificador-decodificador con códigos GRS. Veamos

los detalles del código. La función principal es $CifradoDescifrado$, que es una simulación del funcionamiento completo del criptosistema, generándose las claves, cifrando y descifrando. Recibe como parámetros los mismos 6 que pasa el usuario por línea de comandos. Tras convertir los coeficientes del polinomio base (en caso de que lo hubiera) a un polinomio de Sage, llamamos la función *Gen*. Va a ser la función encargada de generar las claves pública y privada, recibiendo como parámetros todos los entregados por el usuario a excepción del mensaje. Una vez construido el cuerpo finito sobre el que vamos a trabajar (aunque en el constructor de la clase GRS ya lo hacemos, lo necesitamos antes de llamar a dicho constructor para generar los vectores α y v), generamos aleatoriamente el vector v (teniendo en cuenta que no puede tener componentes nulas) y α (recordando que no puede tener entradas repetidas). Podemos ya, por tanto, crear una instancia de la clase GRS, pasando los parámetros $alphaCuerpo$ y $vCuerpo$ a true, ya que todas sus componentes son ya elementos del cuerpo en cuestión y no necesitan procesamiento adicional. Generamos, también de forma aleatoria, la matriz invertible S y la de permutación P, de manera que la clave pública consistirá del producto de las matrices S, la generadora de la instancia de GRS creada antes y P, así como del cuerpo sobre el que trabajamos y el orden del subcuerpo primo, en caso de que lo hubiera, mientras que la clave privada estará formada por la instancia de GRS y las matrices S y P. Una vez *Gen* nos devuelve las claves, generaremos aleatoriamente un texto plano de longitud la dimensión del código, en caso de que se haya pasado un texto plano nulo como parámetro (será así, como veremos, en la llamada a $CifradoDescifrado$ desde *validacionMcElieceGRS.py*, para evitar que una generación del texto plano desde fuera pueda no ser reconocida como del cuerpo sobre el que funciona el código). Podemos ya llamar a la función de cifrado *Enc*, que recibe como parámetros la clave pública, un texto plano y un bit que indica si los elementos de texto plano están en el cuerpo o de lo contrario necesitan procesamiento (el primer caso será cuando generamos el texto plano en $CifradoDescifrado$, mientras que el segundo será cuando pasamos los elementos por línea de comandos como hemos descrito). Dentro de la función, lo primero que hacemos es desglosar la clave pública en sus diferentes elementos, así como obtener la dimensión y longitud del código al saber la relación de ambas con la matriz generadora de la clave pública. Necesitamos estos dos valores para conocer hasta cuántos errores podemos introducir al cifrar el texto plano. Dicho número se genera de manera aleatoria. Finalmente, introducimos dicho número de errores sobre la palabra código resultante de multiplicar el texto plano por la matriz generadora de la clave pública, usando de nuevo las funciones de la clase GRS. Para terminar la simulación, llamamos a la función *Dec* encargada de descifrar el texto devuelto por *Enc*. Recibe dos parámetros, clave privada y texto a descifrar. Una vez desglosada la clave privada, en primer lugar aplicamos el método corregirCodigo de la clase GRS sobre el producto del texto cifrado y la inversa de P, luego decodificamos el resultado con el método decodificar de la misma clase y finalmente multiplicamos el resultado devuelto por el último método por la inversa de la matriz S. La función $CifradoDescifrado$ devuelve el texto cifrado devuelto por *Enc*, el texto plano devuelto por *Dec* y el texto plano pasado a *Enc*, ya sea el que envía el usuario o el generado dentro de la función.

Con el fin de validar el correcto funcionamiento de la función $CifradoDescifrado$, utilizamos el código definido en *validacionMcElieceGRS.py*. Su ejecución es similar a la vista para el archivo *validacionEncDecGRS.py*. Para cada combinación de parámetros, vamos a medir igualmente el número de éxitos y el tiempo medio de ejecución para las iteraciones solicitadas, entendiendo un éxito como aquellos casos donde el texto plano generado aleatoriamente y el devuelto tras descifrar coincidan. En la tabla 5.1 se recogen los

resultados de la ejecución de *validacionMcElieceGRS.py* con 30 iteraciones. Dicha ejecución se puede visualizar en la figura 5.1 y 5.2.

Sin embargo, en la siguiente y última sección del trabajo, vamos a tratar de probar por qué el uso de códigos RS generalizados puede no ser la mejor opción en los criptosistemas basados en teoría de códigos.

Longitud	Orden	Dimensión	Iteraciones	Nº Éxitos	% Éxitos	Tiempo Medio (Seg)
20	81	8	30	30	100	0.049035
60	127	28	30	30	100	0.526065
80	149	30	30	30	100	1.027802
96	193	45	30	30	100	4.975933
128	251	64	30	30	100	24.681221
160	257	80	30	30	100	77.625515
200	281	100	30	30	100	146.230902
240	293	120	30	30	100	338.35086

Tabla 5.1: Resultados simulación McEliece para 30 iteraciones con diferentes parámetros

```
higinio@higinio-PS42-Modern-8RC:~/Plantilla-TFG-master/code$ python3 validacionMcElieceGRS.py --numIteraciones 30
Longitud del código: 20
Orden del cuerpo: 81
Polinomio base: 2*x^4 + x^3 + 1
Orden Subcuerpo primo: 3
Dimensión del código: 8
Número de iteraciones totales: 30
Número de éxitos: 30
Porcentaje de aciertos: 100.0
Tiempo medio de ejecución: 0.049035 segundos

Longitud del código: 60
Orden del cuerpo: 127
Dimensión del código: 28
Número de iteraciones totales: 30
Número de éxitos: 30
Porcentaje de aciertos: 100.0
Tiempo medio de ejecución: 0.526065 segundos

Longitud del código: 80
Orden del cuerpo: 149
Dimensión del código: 30
Número de iteraciones totales: 30
Número de éxitos: 30
Porcentaje de aciertos: 100.0
Tiempo medio de ejecución: 1.027802 segundos

Longitud del código: 96
Orden del cuerpo: 193
Dimensión del código: 45
Número de iteraciones totales: 30
Número de éxitos: 30
Porcentaje de aciertos: 100.0
Tiempo medio de ejecución: 4.975933 segundos

Longitud del código: 128
Orden del cuerpo: 251
Dimensión del código: 64
Número de iteraciones totales: 30
Número de éxitos: 30
Porcentaje de aciertos: 100.0
Tiempo medio de ejecución: 24.681221 segundos

Longitud del código: 160
Orden del cuerpo: 257
Dimensión del código: 80
Número de iteraciones totales: 30
Número de éxitos: 30
Porcentaje de aciertos: 100.0
Tiempo medio de ejecución: 77.625515 segundos
```

Figura 5.1: Resultados simulación McEliece para 30 iteraciones con diferentes parámetros

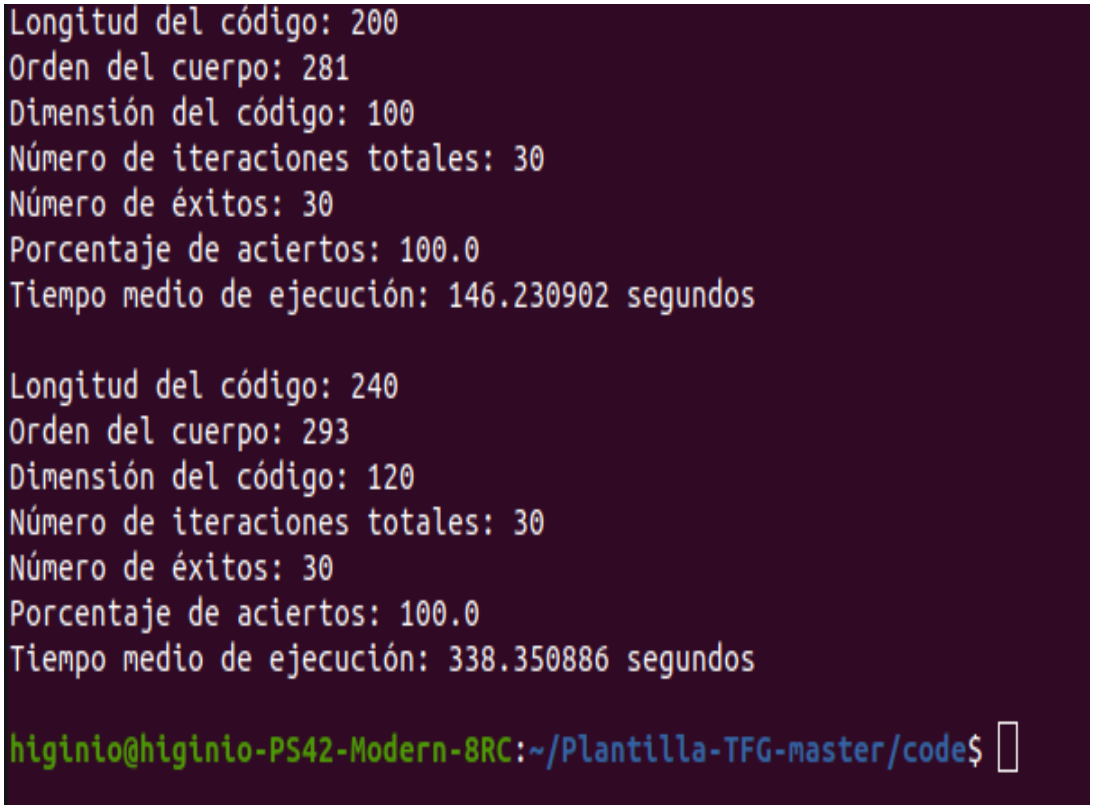

Figura 5.2: Continuación resultados figura 5.1

6 Ataque de Sidelnikov-Shestakov

En este capítulo final del trabajo vamos a estudiar el ataque de Sidelnikov-Shestakov, una propuesta realizada en 1992 por V.M. Sidelnikov y S.O. Shestakov en la que plantean la posibilidad de recuperar los parámetros de un código RS Generalizado conocido el producto de una matriz generadora suya por otra matriz invertible cualquiera. Revisaremos las bases teóricas sobre las que se fundamenta dicho ataque, así como propondremos una posible implementación. Nos basaremos en el contenido de [ySS92] y [Wie10].

Sea G una matriz generadora del código $GRS_{n,k}(\alpha, v)$ y S una matriz invertible, ambas sobre $\mathbb{F}_q$. Definimos su producto $B := SG$. Este va a ser el punto de partida del ataque. Sin embargo, no habría ningún problema en añadir una multiplicación a la derecha por una matriz de permutación P, al igual que en el criptosistema de McEliece, ya que lo que va a realizar esta matriz es cambiar el orden de las componentes de los vectores α y v, pero como dicho cambio es igualmente aplicado al texto cifrado, descifrará al texto plano original, aunque no conozcamos explícitamente la permutación realizada.

Como preparación al ataque, vamos a demostrar la siguiente proposición, que, en esencia, lo que nos viene a decir es que un código RS generalizado está definido de manera única por sus vectores α y v, salvo transformación lineal de α y multiplicación por escalar de v.

Proposición 6.0.1. *Sean $\alpha = (\alpha_1, ..., \alpha_n) \in \mathbb{F}_q^n, v = (v_1, ..., v_n) \in (\mathbb{F}_q^*)^n$, con $\alpha_i \neq \alpha_j$, para $i \neq j$, $i, j = 1, .., n$ y $0 \leq k \leq n$. Se verifica*

$$GRS_{n,k}(\alpha, v) = GRS_{n,k}((a\alpha_1 + b, ..., a\alpha_n + b), (cv_1, ..., cv_n)),$$

$\forall a, b, c \in \mathbb{F}_q, a, c \neq 0$.

Demostración. Sea $(v_1 f(\alpha_1), ..., v_n f(\alpha_n)) \in GRS_{n,k}(\alpha, v)$, para cierto $f(x) \in \mathcal{P}_k$, donde $\mathcal{P}_k$ denota los polinomios de grado menor que k sobre $\mathbb{F}_q[x]$. Si definimos el polinomio $g(x) = \frac{1}{c} f(\frac{x-b}{a})$ (el cual está bien definido ya que $a, c \neq 0$), obtenemos que $(cv_1 g(a\alpha_1 + b), ..., cv_n g(a\alpha_n + b)) = (v_1 f(\alpha_1), ..., v_n f(\alpha_n)) \in GRS_{n,k}((a\alpha_1 + b, ..., a\alpha_n + b), (cv_1, ..., cv_n))$. Sea ahora $(cv_1 f(a\alpha_1 + b), ..., cv_n f(a\alpha_n + b)) \in GRS_{n,k}((a\alpha_1 + b, ..., a\alpha_n + b), (cv_1, ..., cv_n))$, para cierto $f(x) \in \mathcal{P}_k$. Definiendo el polinomio $g(x) = cf(ax + b)$, obtenemos que $(v_1 g(\alpha_1), .., v_n g(\alpha_n)) = (cv_1 f(a\alpha_1 + b), ..., cv_n f(a\alpha_n + b)) \in GRS_{n,k}(\alpha, v)$, probando así la doble inclusión. □

Antes de comenzar la explicación del ataque, cabe mencionar que todos los subíndices que usemos en la formulación matemática del mismo se verán posteriormente decrementados en uno en la implementación en código.

En primer lugar, vamos a calcular una matriz escalonada por filas $E(B)$ de B:

$$E(B) = \begin{pmatrix} 1 & 0 & \dots & 0 & b_{1,k+1} & \dots & b_{1,n} \\ 0 & 1 & \dots & 0 & b_{2,k+1} & \dots & b_{2,n} \\ & & \ddots & & \vdots & & \vdots \\ 0 & \dots & 0 & 1 & b_{k,k+1} & \dots & b_{k,n} \end{pmatrix},$$

para ciertos $b_{i,j}$, con $i = 1, ..., k$, $j = 1, ..., n$. Como la matriz S es invertible, al multiplicar G por ella lo que estamos haciendo es un cambio de base, de manera que B sigue generando el mismo código que G. De igual forma, como $E(B)$ se obtiene a partir de operaciones por filas de la matriz B, el código generado por ella permanece intacto. Por lo tanto, la i-ésima fila de $E(B)$ b_i tiene la forma $(v_1 f_{b_i}(\alpha_1), ..., v_n f_{b_i}(\alpha_n))$, para cierto $f_{b_i}(x) \in \mathcal{P}_k$. Como las columnas $1, ..., i-1$ y $i+1, ..., k$ de b_i son iguales a cero, siendo $k-1$, y el polinomio f_{b_i} tiene grado como mucho $k-1$, tenemos que las componentes de α correspondientes a esas columnas serán las únicas raíces de f_{b_i}, pues además todas las componentes de v son no nulas:

$$f_{b_i}(x) = c_{b_i} \cdot \prod_{j=1, j\neq i}^{k} (x - \alpha_j),$$

con $c_{b_i} \in \mathbb{F}_q \setminus \{0\}$. Escogemos ahora dos filas de $E(B)$, por ejemplo b_1 y b_2, y dividimos las componentes de la primera por las de la segunda, obteniendo

$$\frac{b_{1,j}}{b_{2,j}} = \frac{v_j \dot{f}_{b_1}(\alpha_j)}{v_j f_{b_2}(\alpha_j)} = \frac{c_{b_1}(\alpha_j - \alpha_2)}{c_{b_2}(\alpha_j - \alpha_1)}, \tag{6.1}$$

para $j = k+1, ..., n$. No podemos dividir por 0, pero como $E(B)$ es una matriz generadora del código $GRS_{n,k}(\alpha, v)$, que tiene peso mínimo $n-k+1$, y en cada fila hay un 1, $n-k$ elementos a la derecha y el resto son 0s, necesariamente
$\forall i \in \{1, ..., k\}, \forall j \in \{k+1, ..., n\}, b_{i,j} \neq 0$. Usando la proposición 6.0.1, podemos elegir libremente $\alpha_1 = 0$ y $\alpha_2 = 1$ sin que ello afecte al código que se genera. Esta parte del ataque es la única donde va a ser necesaria la fuerza bruta para encontrar el valor del cociente $c := \frac{c_{b_1}}{c_{b_2}}$, de manera que, en principio, deberemos iterar por todos los elementos no nulos de $\mathbb{F}_q$. Si definimos también $d_j := \frac{b_{1,j}}{b_{2,j}}$ y sustituimos α_1 y α_2, llegamos a que el valor de cada α_j es $\alpha_j = \frac{c}{c - d_j}$. Esta igualdad nos sugiere que en la fuerza bruta podemos descartar los d_j, reduciendo así los posibles candidatos. Queda por tanto determinar los αs restantes, $\alpha_3, ..., \alpha_k$. Si reemplazamos en la ecuación anterior b_2 por b_i, para $i = 3, ..., k$, obtenemos

$$\frac{b_{1,j}}{b_{i,j}} = \frac{c_{b_1}(\alpha_j - \alpha_i)}{c_{b_i}(\alpha_j - \alpha_1)}.$$

Las incógnitas son $\frac{c_{b_1}}{c_{b_i}}$ y α_i. Escogiendo dos valores de j en $j = k+1, ..., n$, por ejemplo $j = k+1, k+2$, y declarando las dos nuevas incógnitas $c_i := \frac{c_{b_1}}{c_{b_i}}$ y $s_i := c_i * \alpha_i$, así como el valor conocido $d_{i,j} = \frac{b_{1,j}}{b_{i,j}}$, logramos un sistema lineal de dos ecuaciones para cada i:

$$\begin{cases} c_i \alpha_{k+1} - s_i = d_{i,k+1} \alpha_{k+1} \\ c_i \alpha_{k+2} - s_i = d_{i,k+2} \alpha_{k+2} \end{cases} \tag{6.2}$$

Obtenemos los α_is simplemente dividiendo s_i por c_i para cada i. La primera parte de la

recuperación del vector α tiene una complejidad en tiempo de $O(qnlog(q))$, ya que una inversión tiene complejidad $O(log(q))$, realizándose $n-k \approx n$ de ellas, y haciendo fuerza bruta sobre un espacio de $q-1-(n-k) \approx q$ elementos, recordando que $k \leq n \leq q$. Por otro lado, la segunda parte tiene complejidad $O(qklog(q))$, ya que resolver el sistema tiene complejidad $O(log(q))$, resolviéndose $k-3+1 \approx k$ de ellos, para $q-1-(n-k) \approx q$ candidatos a vector α. Podemos concluir que la complejidad de la recuperación de α es $O(qnlog(q))$.
Una vez recuperado α, pasemos a la recuperación de v. En primer lugar, debemos encontrar una solución no trivial $c = (c_1, ..., c_{k+1})$ del siguiente sistema lineal:

$$B' \cdot c = 0,$$

donde B' es la matriz formada por las $k+1$ primeras columnas de B. Análogamente, definimos la matriz G' con respecto a G. Se da la igualdad $B' = SG'$, luego, sustituyendo en la última ecuación y multiplicando por S^{-1} en ambos términos (lo cual es posible al ser invertible), podemos reformular el sistema como

$$G' \cdot c = 0.$$

Intercambiando las componentes de v presentes en G', desconocidas, por las componentes de c, conocidas, obtenemos el sistema

$$\begin{pmatrix} c_1\alpha_1^0 & c_2\alpha_2^0 & \dots & c_{k+1}\alpha_{k+1}^0 \\ c_1\alpha_1^1 & c_2\alpha_2^1 & \dots & c_{k+1}\alpha_{k+1}^1 \\ \vdots & \ddots & & \vdots \\ c_1\alpha_1^{k-1} & c_2\alpha_2^{k-1} & \dots & c_{k+1}\alpha_{k+1}^{k-1} \end{pmatrix} \begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_{k+1} \end{pmatrix} = 0. \tag{6.3}$$

Se trata de un sistema compatible indeterminado (ya que tenemos menos ecuaciones que incógnitas) con infinitas soluciones, pero de nuevo, por la proposición 6.0.1, podemos escoger libremente una de las componentes de v sin alterar el código resultante, por ejemplo $v_1 = 1$, de manera que la solución pasa a ser única. En consecuencia, ya conocemos completamente la matriz G', pues tenemos el vector α completo y las $k+1$ primeras componentes de v. Sea ahora G'' la matriz formada por las primeras k columnas de G, y B'' la matriz análoga para B. Se verifica que $S = B''(G'')^{-1}$ (G'' tiene inversa ya que las filas de G son linealmente independientes por constituir una base, luego lo serán también las k primeras componentes de cada una de ellas, por lo que G'' tiene determinante no nulo). Una vez determinada S podemos calcular $G = S^{-1}B$, y como sabemos que la primera fila de G son las componentes del vector v, podemos reconstruir el resto de dicho vector. En cuanto a la complejidad de la recuperación de v, tenemos que es $O(qk^3)$, el término q procedente de la iteración por los candidatos a α, y k^3 procedente de las inversiones de matrices de orden k, que prevalecen sobre las otras operaciones de resolución de sistemas. Una precondición del ataque es que debe verificarse $2 \leq k \leq n-2$ (necesitamos al menos dos filas para (6.1) , y una vez calculada la primera parte del vector α, necesitamos al menos dos componentes de dicho vector a partir del índice $k+1$ para (6.2)).

Las demostración teórica del ataque se plasma en código en el archivo *SidelnikovShestakov.py*. Necesitaremos tener instalados en nuestro equipo tanto Python como SageMath para poder ejecutarlo. Requiere de 5 parámetros para su ejecución, *longitud*, *ordenCuerpo*, *polinomioBase*, *ordenSubcuerpoPrimo* y *dimension*, siguiendo las mismas

reglas que los parámetros con el mismo nombre del archivo *encDecGRS.py*. Veamos el funcionamiento del código. Tras el procesamiento de los parámetros mencionados, llamamos a la función *ataqueSidelnikovShestakov*, a la que pasamos los cinco parámetros. He agrupado todo el código del ataque dentro de dicha función ya que luego la necesitaremos llamar desde el código de validación del ataque, que presentamos más adelante. Dentro de la función, lo primero que hacemos es verificar que los parámetros cumplen la condición comentada anteriormente, $2 \leq k \leq n-2$, pues si no, sabemos que el ataque no tendrá éxito. En caso de que el usuario haya pasado polinomio irreducible y cuerpo primo para construir un cuerpo no primo, convertiremos la lista de coeficientes a un elemento de anillo de polinomios de Sage y construimos el cuerpo deseado. Una vez realizado este preprocesamiento, pasamos ya al ataque. Para demostrar su efectividad, partimos de un texto plano aleatorio, generamos clave pública y privada (aunque esta última no nos servirá) de criptosistema de McEliece usando los parámetros dados, ciframos el texto plano y nos quedamos con la matriz de la clave pública, que es el punto de partida del ataque. Calculamos una forma escalonada suya y a partir de ella podemos obtener los cocientes $\frac{b_{1,j}}{b_{2,j}}$ para $j = k+1, ..., n$. A partir de la ecuación que hemos deducido anteriormente, $\alpha_j = \frac{c}{c-d_j}$ (donde recordemos que $d_j = \frac{b_{1,j}}{b_{2,j}}$) y realizando fuerza bruta sobre c, descartando el 0 y los d_j (en el código la variable equivalente a d_j será b_j), obtenemos los primeros candidatos a α, siendo cada candidato una lista de $n-k$ elementos. Como por definición de código GRS el vector α no puede contener elementos repetidos, usamos dicho criterio como primer filtrado de candidatos. Antes de pasar al cálculo del resto de posiciones de los candidatos, realizamos una copia de dichos candidatos e insertamos sobre dicha copia $\alpha_1 = 0$ y $\alpha_2 = 1$, que eran fijos, al comienzo de todos ellos. Las razones de realizar dicha copia son dos: por un lado, como α_{k+1} y α_{k+2} son los que vamos a utilizar para resolver los sistemas de ecuaciones y se corresponden con las posiciones 0 y 1 de todos los candidatos, al añadir α_1 y α_2 en dichas posiciones, pero sobre las copias, evitamos que se usen los valores erróneos. Por otro lado, una vez que obtengamos los α_is, para $i = 3, ..., k$, podemos añadir cada uno de ellos directamente a la posición de las copias de los candidatos que coincide con su índice, evitando así cálculos extras para dicho índice. Veamos ahora la función *calcularAlphas*. Recibe como parámetros la dimensión del código k ($j = k+1, k+2$ son los que utilizamos para resolver los sistemas), la matriz pública escalonada EB (necesaria para calcular los $d_{i,j} = \frac{b_{1,j}}{b_{i,j}}$, *indice* (representa el índice i del $alpha_i$ que queremos obtener al resolver el sistema), *candidatosAlpha* (lista de candidatos, no pasamos la copia sino la primera versión) y $field$ (cuerpo sobre el que estamos trabajando, necesario para inicializar las matrices y vectores que definen el sistema). Para cada candidato a α, resolvemos el sistema planteado antes y deshacemos el cambio de variable propuesto para obtener la componente de α dada por *indice*, siempre que la incógnita c_i ($soluciones[0]$ en el código) no sea nula. Si la componente de α obtenida estaba ya en el candidato en cuestión, la ignoramos. Finalmente, la función devuelve la lista de todos los α_{indice} para cada candidato, devolviendo *None* para aquellos que no se han podido calcular o se han ignorado. A dicha función la llamamos dentro de un bucle que itera desde 2 hasta $k-1$ (se pasarán como índice), calculando por tanto para todos los candidatos sus componentes restantes. Para terminar el cálculo de los candidatos a α, de nuevo descartamos aquellos con componentes repetidas, o que tengan alguna posición nula, resultado de la función *calcularAlphas*. Pasamos ahora a la obtención de los candidatos a v. En primer lugar, definimos la matriz $B1$ (la equivalente en el código a la matriz B'), y calculamos una base de su núcleo, equivalente a resolver el sistema

compatible indeterminado $B' \cdot c = 0$. Comenzamos de nuevo un recorrido por los candidatos a α obtenidos definiendo la matriz $G1$ (la equivalente a la matriz del sistema (6.3)), con cuyo núcleo obtendremos las $k+1$ primeras componentes de los candidatos a v. Sobreescribimos el contenido de $G1$ con la equivalente a la matriz G', que calculamos usando las componentes conocidas para los candidatos a v, junto con las correspondientes en los candidatos a α. Con $G1$ y B, podemos calcular las candidatas a matriz S, y una vez obtenida, determinamos enteramente las candidatas a G. Usando la primera fila de las candidatas a G, resolvemos el resto de componentes de los candidatos a v. De nuevo por la definición de código GRS, descartamos aquellos candidatos a v que tengan alguna posición nula. Estamos ya en condiciones de definir un código GRS que nos permita descifrar la variable *ct*. Para ello, usamos primero la función de la clase GRS *corregirCodigo*, una vez eliminados los errores introducidos llamamos a la función de la misma clase decodificar, y finalmente multiplicamos el resultado a la derecha por la inversa de la candidata a matriz S. Una vez hayamos obtenido un candidato a texto plano, salimos del bucle para evitar iteraciones innecesarias. La función *ataqueSidelnikovShestakov* devolverá el texto plano original generado y el candidato obtenido como resultado del ataque.

Con el fin de validar el correcto funcionamiento de la función *ataqueSidelnikovShestakov*, utilizamos el código definido en *validacionSidelnikovShestakov.py*. Se ejecuta de manera análoga al archivo *validacionEncDecGRS.py*. Para cada combinación de parámetros, vamos a medir el número de éxitos del ataque y el tiempo medio de ejecución para las iteraciones solicitadas, entendiendo un éxito como aquellos casos donde el texto plano original y el candidato coincidan. En la tabla 6.1 se recogen los resultados de la ejecución de *validacionSidelnikovShestakov.py* con 30 iteraciones. Dicha ejecución se puede visualizar en la figura 6.1 y 6.2.

Longitud	Orden	Dimensión	Iteraciones	Nº Éxitos	% Éxitos	Tiempo Medio (Seg)
20	81	8	30	30	100	0.111893
60	127	28	30	30	100	0.814900
80	149	30	30	30	100	1.017084
96	193	45	30	30	100	3.841444
128	251	64	30	30	100	29.891841
160	257	80	30	30	100	45.649180
200	201	100	30	30	100	224.820442
240	293	120	30	30	100	358.243859

Tabla 6.1: Resultados ataque Sidelnikov-Shestakov para 30 iteraciones con diferentes parámetros

```
higinio@higinio-PS42-Modern-8RC:~/Plantilla-TFG-master/code$ python3 validacionSidelnikovShestakov.py --numIteraciones 30
Longitud del código: 20
Orden del cuerpo: 81
Polinomio base: 2*x^4 + x^3 + 1
Subcuerpo primo: 3
Dimensión del código: 8
Número de iteraciones totales: 30
Número de éxitos: 30
Porcentaje de aciertos: 100.0
Tiempo medio de ejecución: 0.111893 segundos

Longitud del código: 60
Orden del cuerpo: 127
Dimensión del código: 28
Número de iteraciones totales: 30
Número de éxitos: 30
Porcentaje de aciertos: 100.0
Tiempo medio de ejecución: 0.814900 segundos

Longitud del código: 80
Orden del cuerpo: 149
Dimensión del código: 30
Número de iteraciones totales: 30
Número de éxitos: 30
Porcentaje de aciertos: 100.0
Tiempo medio de ejecución: 1.017084 segundos

Longitud del código: 96
Orden del cuerpo: 193
Dimensión del código: 45
Número de iteraciones totales: 30
Número de éxitos: 30
Porcentaje de aciertos: 100.0
T Texmaker io de ejecución: 3.841444 segundos

Longitud del código: 128
Orden del cuerpo: 251
Dimensión del código: 64
Número de iteraciones totales: 30
Número de éxitos: 30
Porcentaje de aciertos: 100.0
Tiempo medio de ejecución: 29.891841 segundos

Longitud del código: 160
Orden del cuerpo: 257
Dimensión del código: 80
Número de iteraciones totales: 30
Número de éxitos: 30
Porcentaje de aciertos: 100.0
Tiempo medio de ejecución: 45.649180 segundos
```

Figura 6.1: Resultados ejecución ataque Sidelnikov-Shestakov para 30 iteraciones con diferentes parámetros

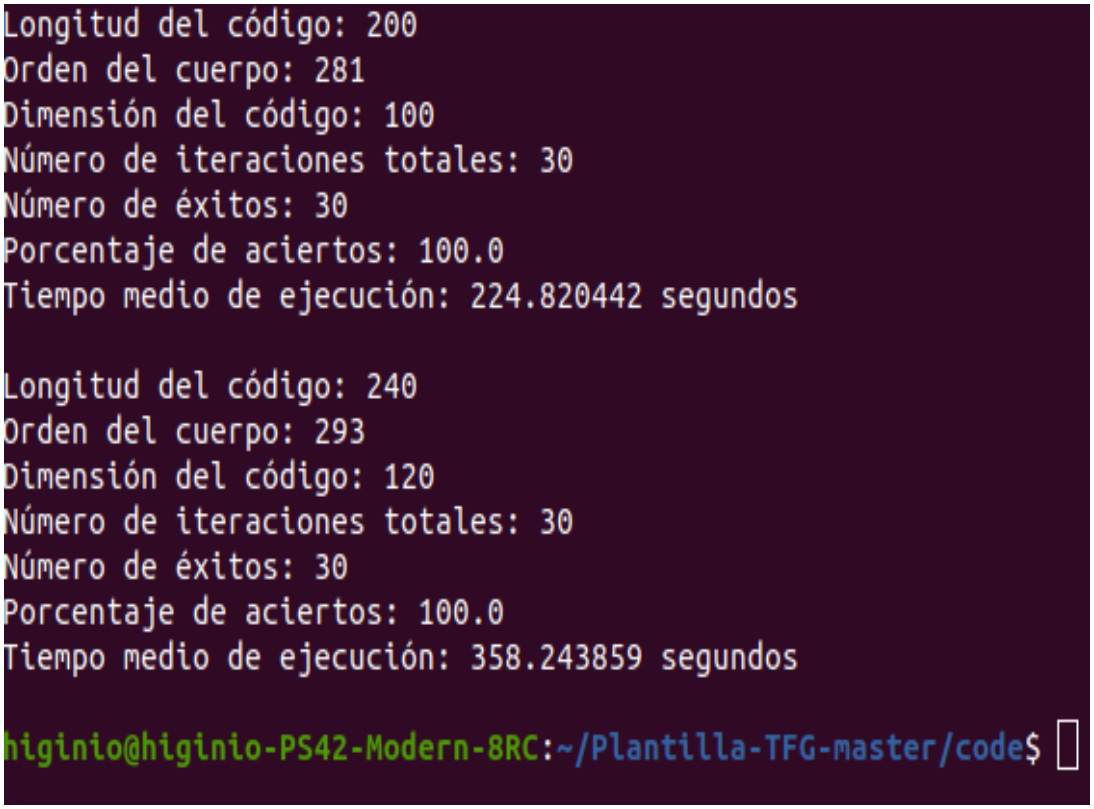

Figura 6.2: Continuación ejecución figura 6.1

Conclusiones y Trabajos Futuros

Considero que el criptosistema de McEliece es una propuesta muy interesante dentro de la criptografía post-cuántica, especialmente debido a su facilidad de implementación, una vez conocidas las nociones básicas sobre teoría de códigos. Como principal desventaja destacaría los elevados tiempos de ejecución que requiere para el cifrado-descifrado cuando el número de parámetros crece, como hemos visto durante la presentación de los resultados de la experimentación. Como solución, dedicaría un esfuerzo adicional para la búsqueda e implementación de un algoritmo de inversión de matrices más eficiente, ya que es claramente el cálculo más demandante durante el descifrado. Por otro lado, la efectividad demostrada teórica y empíricamente del ataque de Sidelnikov-Shestakov nos obliga a descartar a los códigos Reed-Solomon generalizados como alternativa a los Goppa. Por ello, como posible continuación de este estudio sería interesante probar el criptosistema con otras familias de códigos que sigan ofreciéndonos tamaños de claves razonables, pero con una menor estructuración que los Reed-Solomon generalizados, donde radica su vulnerabilidad.

Bibliografía

[D.E07] R.Overbeck y A.Schmidt D.Engelbert. A summary of mceliece-type cryptosystems and their security. https://www.researchgate.net/publication/220336630_A_Summary_of_McEliece-Type_Cryptosystems_and_their_Security, 2007.

[ERB78] Henk C.A Van Tilborg Elwyn R. Berlekamp, Robert J. McEliece. On the inherent intractability of certain coding problems. *IEEE Transactions on Information Theory*, 24(3), 1978.

[Exca] Math Stack Exchange. abstract algebra - show that $x^n - 1$ has no repeated irreducible factors over f_q if and olnly if $gcd(q, n) = 1$. https://math.stackexchange.com/questions/4110755/show-that-xn-1-has-no-repeated-irreducible-factors-over-mathbbf-q-if-and.

[Excb] Math Stack Exchange. Explain why the hamming distance satisfies the triangle inequality. https://math.stackexchange.com/questions/555180/explain-why-the-hamming-distance-satisfies-the-triangle-inequality.

[Hala] J.I. Hall. Notes on coding theory. chapter 5. generalized reed-solomon codes. https://users.math.msu.edu/users/halljo/classes/CODENOTES/GRS.pdf.

[Halb] J.I. Hall. Notes on coding theory. chapter a.3. special topics. https://users.math.msu.edu/users/halljo/classes/CODENOTES/SpecAlg.pdf.

[Mor20] Serafín Moral. Tema 1: Máquinas de turing. funciones y lenguajes calculables, Febrero 2020. Material de la asignatura Modelos de Computación, Universidad de Granada.

[Val] Ashley Valentijn. Goppa codes and their use in the mceliece cryptosystems (2015). syracuse university honors program capstone projects. 845. https://surface.syr.edu/honors_capstone/845.

[Vau23] Serge Vaudenay. Cryptography and security lecture notes, 2023. Material de la asignatura Cryptography and Security, EPFL, Lausanne, Suiza.

[Wie10] Christian Wieschebrink. Cryptanalysis of the niederreiter public key scheme based on grs subcodes. https://www.researchgate.net/publication/220961286_Cryptanalysis_of_the_Niederreiter_Public_Key_Scheme_Based_on_GRS_Subcodes, 2010.

[Wik] Wikipedia. Lagrange polynomial - wikipedia. https://en.wikipedia.org/wiki/Lagrange_polynomial.

[ySS92] V.M. Sidelnikov y S.O Shestakov. On insecurity of cryptosystems based on generalized reed-solomon codes, 1992.

[yVP03] W.Cary Huffman y Vera Pless. *Fundamentals of Error Correcting Codes*. Cambridge University Press, primera edición, 2003.