Sección de Matemáticas
Universidad de La Laguna

Manuel Ramos Montesó

Estudio e Implementación del Criptosistema de McEliece con SageMath

Study and Implementation of the McEliece Cryptosystem with SageMath

Trabajo Fin de Grado
Grado en Matemáticas
La Laguna, Septiembre de 2022

Dirigido por
Pino Caballero Gil

Pino Caballero Gil
Ingeniería Informática y de Sistemas
Universidad de La Laguna
38200 La Laguna, Tenerife

Agradecimientos

Agradecer a todos los profesores que me han ayudado a lo largo de mis estudios universitarios y en especial a la tutora Pino Caballero Gil.
También agradecer a mi familia y amigos por el apoyo incondicional durante esta etapa.

Manuel Ramos Montesó
La Laguna, 8 de septiembre de 2022

Resumen · Abstract

Resumen

La criptografía que usamos en la actualidad será vulnerable tras el desarrollo y despliegue de los ordenadores cuánticos, debido a que ya se conocen varios algoritmos cuánticos capaces de romper los actuales criptosistemas en un tiempo polinomial. De hecho, ahora mismo la comunidad científica está estudiando a contrarreloj cómo poder resistir dichos ataques para poder seguir garantizando la ciberseguridad.

Este Trabajo Fin de Grado se centra en el estudio e implementación del criptosistema de McEliece, el cual parece resistente a los ordenadores cuánticos. Dicho criptosistema se basa en los Códigos de Goppa, que son códigos lineales.

En los tres primeros capítulos se introducen las bases del criptosistema de McEliece. Los capítulos cuarto y quinto se centran en su estudio e implementación en SageMath, reflejando la gran complejidad que conllevaría romper el criptosistema.

Palabras clave: *Criptografía Postcuática – Código Lineal – Criptosistema – McEliece*

Abstract

The cryptography we use today will become vulnerable with the development and deployment of quantum computers, given that several quantum algorithms capable of breaking cryptosystems in polynomial time are already known. In fact, right now the scientific community is studying against time how to resist such attacks in order to continue guaranteeing cybersecurity.

This Final Degree Project focuses on the study and implementation of the McEliece cryptosystem, which seems resistant to quantum computers. This cryptosystem is based on Goppa Codes, which are linear codes.

In the first three chapters, the bases of the McEliece cryptosystem are introduced. The fourth and fifth chapters focus on its study and implementation in SageMath, reflecting the great complexity that breaking the cryptosystem would require.

Keywords: *post-quantum cryptography – linear code – cryptosystem – McEliece .*

Contenido

1

INTRODUCCIÓN

1.1. Motivación

A lo largo de la historia de la humanidad han sido muchos los métodos usados para cifrar información, la mayoría basados en matemáticas. Los primeros métodos se basaban en criptografía de clave secreta en la que un emisor y un receptor se intercambiaban una clave con la que creaban un canal seguro para la comunicación. Entre ellos podemos destacar el cifrado de César que data del siglo I a.C, el Criptosistema afín que es una mejora del cifrado César, cifrados de sustitución, o los más actuales como DES. Para incrementar la seguridad en el siglo XX surgió la criptografía de clave pública, en la que el receptor es el único que tiene acceso a su clave privada. Los criptosistemas de clave pública con el paso del tiempo se han ido sofisticando y adaptando a las nuevas tecnologías pero el paso gigantesco que se dará con el ordenador cuántico pondrá en peligro la seguridad que esos sistemas. Para resolver el problema se están estudiando criptosistemas que resistan ataques cuánticos. El Instituto Nacional de Estándares y Tecnología (NIST, National Institute of Standards and Technology) está llevando a cabo un concurso para seleccionar un estándar de criptografía post-cuántica. El Criptosistema de McEliece se encontraba entre unos de los candidatos y por ello fue escogido como objeto de este Trabajo Fin de Grado.

1.2. Objetivo

El objetivo principal de este Trabajo Fin de Grado es el análisis e implementación del criptosistema de McEliece. De esta forma se pretende comprender por qué es tan resistente a ataques con ordenadores cuánticos. Especialmente se pretende profundizar en su implementación en SageMath ya que esta herramienta incorpora muchos paquetes que hacen posible programar códigos lineales. Sin embargo, ha sido necesario implementar desde cero los códigos de Goppa. A lo largo de los capítulos de este trabajo se muestran diferentes ejemplos para facilitar la comprensión del sistema [1] [2].

2

FUNDAMENTOS MATEMÁTICOS

2.1. Grupos

Definición 2.1.1. *Un grupo es un par (G,$\star$), donde G es un conjunto no vacío y $\star$ es una operación binaria e interna que verifica:*

1. *La operación $\star$ es asociativa.*

2. *La operación $\star$ tiene elemento neutro, esto es, $\exists\ e \in G$ tal que $\forall g \in G$, se tiene*
$$e \star g = g \star e = g$$

3. *Todo elemento $g \in G$ tiene simétrico respecto de $\star$, esto es, $\exists\ g' \in G$ tal que*
$$g \star g' = g' \star g = e$$

4. *Si además*
$$g \star h = h \star g$$
para todo par de elementos $g, h \in g$, se dice que G es abeliano o conmutativo.

Definición 2.1.2. *Si G es finito, se denomina orden de G al entero* $|G| = card(G)$

Definición 2.1.3. *Un subconjunto H no vacío de G se dice que es un subgrupo de $(G, \star)$ si $\star$ es interna en H (es decir, si $H \star H \subseteq H$) y $(H, \star|_{H\times H})$ es un grupo y se denota $H \leq G$*

Proposición 2.1.4. *Sea $(G, \star)$ un grupo y $H \subseteq G$ un subconjunto no vacío. Las condiciones siguientes son equivalentes:*

1. *H es un subgrupo de $(G, \star)$.*
2. *Para todo par de elementos $x, y \in H$ se tiene que $x \star y' \in H$.*

En lo sucesivo, denotaremos la operación $\star$ simplemente como la multiplicación, y ocasionalmente, la suma.

Proposición 2.1.5. *Sea G un grupo y $\{H_i\}_{i\in I}$ una familia de subgrupos. Entonces $\bigcap_{i\in I} H_i$ es un subgrupo de G.*

Definición 2.1.6. *Dado un subconjunto $S \subseteq G$ de un grupo, se llama subgrupo generado por S a*

$$\langle S\rangle = \bigcap_{\substack{H\, subgrupo\, de\, G\\ H\supseteq S}} H$$

Proposición 2.1.7. *Sea S' el conjunto de los inversos de los elementos de $S \subseteq G$. Entonces,*

$$\langle S\rangle = \{x_1 \star x_2 \star \cdots \star x_n,\ tal\ que\ x_i \in S \cup S', y\ n \geq 1\}.$$

Dicho de otra forma, el subgrupo generado por S es el conjunto de todos los productos finitos de elementos de S y de sus inversos.

Definición 2.1.8. *Se llama orden de a al menor entero positivo k tal que $a^k = e$, y se denota $o(a)$.*

Teorema 2.1.9 (Teorema de Lagrange)**.** *Sea G un grupo finito y $H \subseteq G$ un subgrupo. Entonces $|H|$ divide a $|G|$. En particular, $o(a) = |\langle a \rangle|$ divide a $|G|$ para todo elemento $a \in G$.*

Demostración.

Sean $H \leq G$, y definamos la relación de equivalencia

$$x \sim y \Longleftrightarrow x' \star y \in H.$$

En esta situación, la clase de equivalencia de x, que denotamos [x], verifica $[x] = x \star H$.
El conjunto cociente $G/\sim$, que no será en general un grupo, es finito y de la forma

$$G/H = \{x_1 \star H, \ldots, x_r \star H\}.$$

Pero todas las clases $x_i \star H$ tienen el mismo cardinal, porque dado $a \in G$, la aplicación $\mu_a : G \to G$ dada por $x \mapsto xa$ es una biyección. Por tanto,

$$card(H) = card(\mu_{x_i}(H)) = card(x_i \star H),$$

y así, $|G| = r \cdot |H|$.

Definición 2.1.10. *Un grupo G se llama cíclico si existe un $a \in G$ tal que*

$$G = \langle a \rangle = \langle \{a\} \rangle = \{a^m \mid m \in \mathbb{Z}\}$$

Definición 2.1.11. *Decimos que H es normal en G cuando, $\forall x \in G$, se tiene $x \star H = H \star x$. Esta condición se denota habitualmente $H \lhd G$.*

Proposición 2.1.12. *El conjunto cociente, que se denota habitualmente G/H, tiene estructura de grupo con la operación (notada también $\star$):*

$$(x \star H) \star (y \star H) = (x \star y) \star H.$$

Cuando G es abeliano, es inmediato ver que todo subgrupo es normal y, por tanto, siempre podemos considerar la estructura cociente.
Como ejemplo: $\mathbb{Z}/\mathbb{Z}_m$

Definición 2.1.13. *Un homomorfismo de grupos es una aplicación*

$$f : (G, \star) \longrightarrow (L, \odot)$$

que verifica que para cualquier par de elementos $x, y \in G$,

$$f(x \star y) = f(x) \odot f(y),$$

esto es,un homomorfismo es una aplicación que respeta las operaciones de grupo.

2.2. Anillos

Definición 2.2.1. *Un anillo es una terna $(A, +, \cdot)$ formado por un conjunto A y dos operaciones internas y binarias $+,\cdot$ verificando:*

1. *El par $(A, +)$ es un grupo abeliano, cuyo elemento neutro llamaremos normalmente cero, denotado 0_A, o simplemente 0.*

2. *La operación binaria $\cdot$ es asociativa y tiene elemento neutro,que llamaremos normalmente uno, denotado 1_A, o simplemente 1.*

3. *La operación $\cdot$ es distributiva a la derecha y a la izquierda respecto de la operación $+$, i.e. para todos $x, y, z \in A$, se tiene*

$$(x + y) \cdot z = x \cdot z + y \cdot z, \quad x \cdot (y + z) = x \cdot y + x \cdot z.$$

Si además la operación $\cdot$ es conmutativa, diremos que el anillo es conmutativo.

Definición 2.2.2. *Sea A un anillo. Una unidad es un elemento que posee un simétrico multiplicativo (a la izquierda y a la derecha), que llamaremos inverso. El conjunto de las unidades de A es un grupo para el producto y se notará A^*.*

Definición 2.2.3. *Un cuerpo es un anillo conmutativo tal que todo elemento distinto de cero es una unidad, i.e.,* $A^* = A \setminus \{0\}$

Definición 2.2.4. *Sea* A *un anillo. Un elemento* $x \in A$ *se llamará un divisor de cero si y sólo si es distinto de cero y existe* $y \in A$, $y \neq 0$, *tal que* $xy = 0$. *Un anillo sin divisores de cero se llama un dominio de integridad*

Definición 2.2.5. *Un elemento* $x \in A$ *se llamará nilpotente si es distinto de cero y existe un entero* $n > 0$ *tal que* $x^n = 0$.

Definición 2.2.6. *Sea* A *un anillo. Un ideal de* A *es un subconjunto* I *de* A *que verifica:*

1. *I es un subgrupo del grupo aditivo de A.*

2. *Para cualesquiera $a \in I$ y $x \in A$, se tiene $xa \in I$*

Definición 2.2.7. *Un dominio euclídeo es un dominio de integridad* A *junto con una aplicación* $\mu : A \setminus \{0\} \longrightarrow \mathbb{Z}_{\geq 0}$, *llamada la norma euclídea, que verifica*

$$\mu(a) \leq \mu(ab), \forall a, b \in A \setminus \{0\}$$

y tales que dados dos $a, b \in A$ *con* $b \neq 0$, *existen unos* r *y* q, *tales que*

$$a = bq + r, \qquad \text{con } r = 0, \text{ o bien } 0 \leq \mu(r) < \mu(b)$$

Estos elementos r *y* q *se llaman resto y cociente de la división euclídea de* a *por* b.

Definición 2.2.8. *En un dominio euclídeo, si* $a|b$ *y* $b|a$, *necesariamente* $a = ub$, *donde* u *es una unidad. En este caso, se dirá que* a *y* b *son asociados.*
Si $a|b$ *y* a *no es unidad ni asociado de* b, *se dirá que* a *es un divisor propio de* b.

Definición 2.2.9. *Un $p \in A \setminus \{0\}$ no unidad se llama irreducible si $p = p_1 p_2 \Rightarrow p_1$ unidad y p_2 asociado (o al revés).*

Definición 2.2.10. *Sea $p \in A$ un elemento no nulo que no es una unidad. Se dice que p es primo si verifica la siguiente propiedad:*

$$p|(ab) \Longrightarrow p|a \text{ o } p|b.$$

Teorema 2.2.11. *Sea A un dominio:*

1. *Todo elemento primo es irreducible.*

2. *Si A es un dominio euclídeo, todo irreducible es primo.*

Definición 2.2.12. *Sean a, b y $m \neq 0$ tres enteros. Diremos que a es congruente con b módulo m, notado $a \equiv b \ (mod\ m)$, si b-a es divisible por m.*
La congruencia se puede expresar de forma equivalente:

$$\begin{aligned} &a \equiv b \ (mod\ m) \iff \\ &\text{resto de la división de } a \text{ y } b \text{ por } m \text{ coinciden} \iff \\ &\text{existe } k \in \mathbb{Z} \text{ tal que } a = b + km. \end{aligned}$$

Definición 2.2.13. *La relación $a \sim b \Longleftrightarrow a \equiv b \ (mod\ m)$ es de equivalencia. Llamaremos clase de congruencia de a módulo m a la clase de equivalencia de a, que denotamos $a + \mathbb{Z}_m$. El conjunto de todas las clases de equivalencia módulo m se denota $\mathbb{Z}/\mathbb{Z}_m$.*
Está claro que $\mathbb{Z}/\mathbb{Z}_m = \{0 + \mathbb{Z}_m, \ldots, (m-1) + \mathbb{Z}_m\}$.

Corolario 2.2.14. *Las unidades del anillo $\mathbb{Z}/\mathbb{Z}_m$ son exactamente las clases $a + \mathbb{Z}_m$ con $mcd(a, b) = 1$.*

Corolario 2.2.15. *El anillo $\mathbb{Z}/\mathbb{Z}_m$ es un cuerpo si y sólo si m es primo.*

2.3. Cuerpos Finitos

Definición 2.3.1. *Un cuerpo $\mathbb{F}$ es finito si posee un número finito de elementos, donde el número de elementos del cuerpo se dice el orden de F.*
En general, denotaremos por $\mathbb{F}_q$ al cuerpo finito de q elementos.

Proposición 2.3.2. *Todos los cuerpos finitos con el mismo número de elementos son isomorfos entre sí.*

Proposición 2.3.3. *Si p es un número primo, el conjunto de los enteros $\mathbb{Z}$ módulo p forman un cuerpo, denotado como $\mathbb{F}_p = \mathbb{Z}/\mathbb{Z}_p$.*

Teorema 2.3.4. *Sea F un cuerpo finito. Entonces existe un primo $p \in \mathbb{Z}$ tal que $|F| = p^r$. Todo cuerpo con p^r elementos es isomorfo a F (como grupo abeliano). Cualquier elemento de esta familia de cuerpos isomorfos se denota $\mathbb{F}_{p^r}$*

En la práctica, para hacer cálculos con cuerpos finitos, necesitamos una representación de $\mathbb{F}_q$ que sea manejable.
Una opción es la representación polinomial, que proviene directamente de la Teoría de Galois. En efecto, podemos considerar el cuerpo $\mathbb{F}_q$, con $q = p^r$ elementos, dado por

$$\mathbb{F}_q \simeq \mathbb{F}_q[X]/\langle f(X)\rangle$$

donde f (X) es un polinomio irreducible de grado r
Los monomios $1, x, \ldots, x^{r-1}$ forman una base de $\mathbb{F}_q$ como $\mathbb{F}_p$-espacio vectorial. Por tanto cualquier elemento en este cuerpo se representa de manera única por un polinomio $g(x) \in \mathbb{F}_p[x]$ de grado a lo sumo $r - 1$

Teorema 2.3.5. *Dado un cuerpo finito $\mathbb{F}_q$, el grupo de las unidades $(\mathbb{F}_q^\star, \cdot)$ es un grupo cíclico.*
$\mathbb{F}_q^\star = \mathbb{F}_q \setminus \{0\}$ es de orden $q-1$.

Definición 2.3.6. *Un elemento $x \in \mathbb{F}_q^\star$ que genere $\mathbb{F}_q^\star$ como grupo cíclico multiplicativo se denomina un elemento primitivo de $\mathbb{F}_q$*

Proposición 2.3.7. *Sea $\alpha \in \mathbb{F}_q^\star$ un elemento primitivo. Entonces*
$\mathbb{F}_q = \mathbb{F}_p[\alpha] = \mathbb{F}_{p^r}$.
$\mathbb{F}_p[\alpha]$ es la extensión de $\mathbb{F}_p$.

Observación 2.3.8. *Si partimos de un polinomio irreducible $f(x)$ en $\mathbb{F}_p$ de grado r, podemos añadir una raíz α de $f(x)$ a $\mathbb{F}_p$ y obtener el cuerpo $\mathbb{F}_{p^r} \cong \mathbb{F}_p[\alpha]$ de tal manera que todas las raíces de $f(x)$ se encuentren en $\mathbb{F}_{p^r}$.*

Definición 2.3.9. *Sea $\mathbb{E}$ una extensión de cuerpos finita de $\mathbb{F}_p$. Entonces $\mathbb{E}$ es un $\mathbb{F}_p -$ espacio vectorial; es decir, necesariamente $\mathbb{E} = \mathbb{F}_{p^t}$, para algún entero positivo t. Cada elemento $\alpha \in \mathbb{E}$ es una raíz del polinomio x^{p^t}. Por tanto, existe un polinomio mónico $m_\alpha(x)$ en $\mathbb{F}_p[x]$ que tiene a α como raíz con el menor grado posible. A tal polinomio $m_\alpha(x)$ se le denomina el polinomio mínimo de α en $\mathbb{F}_p$.*

3

CÓDIGOS CORRECTORES DE ERRORES

3.1. Conceptos básicos

Definición 3.1.1. *Un alfabeto $\mathcal{A}$ es un conjunto no vacío de símbolos, sin significado individual. Una palabra del código de n caracteres, o un bloque de código de longitud n es un elemento de $\mathcal{A}^n$. Un código es un conjunto no vacío de palabras, es decir es un $\mathcal{C} \subseteq \mathcal{A}^n$*
Dado un código $\mathcal{C} \subseteq \mathcal{A}^n$, se llama longitud del código al entero n, y tamaño del código al entero $|\mathcal{C}|$.

Definición 3.1.2. *Definimos la métrica o distancia de Hamming en $\mathcal{A}^n$ como*

$$\begin{aligned} d : \mathcal{A}^n \times \mathcal{A}^n &\longrightarrow \mathbb{R}_{\geq 0} \\ (x, y) &\longmapsto card\{\, i \mid x_i \neq v_i,\ \ i = 1, \ldots, n \,\} \end{aligned}$$

esto es, $d(x, y)$ mide el número de coordenadas distintas de x y v.

Definición 3.1.3. *Supongamos recibida una palabra z, definimos su decodificación por máxima verosimilitud como la palabra del código x (si existe) tal que x maximiza*

$$P[z \text{ recibida} \mid x \text{ enviada}]$$

Definición 3.1.4. *Si hemos recibido una palabra $z \in \mathcal{A}^n$, definimos su decodificación por distancia mínima a la única palabra x del código (si existe) que minimiza $d(x, z)$, es decir, x será la palabra más próxima a z.*

Definición 3.1.5. *Vamos a definir los parámetros de un código. Supongamos que $|\mathcal{A}| = q$. Dado un código $\mathcal{C} \subseteq \mathcal{A}^n$, llamamos:*

1. *Distancia mínima del código al número*

$$d(\mathcal{C}) = min\{d(x, v) \mid x, v \in \mathcal{C}\}.$$

2. *Distancia mínima relativa del código al número*

$$\delta(\mathcal{C}) = \frac{d(\mathcal{C})}{n}$$

3. *Tasa de transmisión de información del código al número*

$$R(\mathcal{C}) = \frac{log_q(|\mathcal{C}|)}{n}$$

4. *Redundancia del código al número*

$$n - log_q(|\mathcal{C}|)$$

Normalmente diremos que $\mathcal{C}$ es un código tipo (n, m, d) para decir que es un código de longitud n, tamaño m y distancia mínima d.

Lema 3.1.6. *Sea $\mathcal{C}$ un código tipo (n, m, d). Entonces, para todos $x \in \mathcal{C}$ y $v \in \mathcal{A}^n$, si $d(x, v) < d$, se tiene que $v \notin C$.*
Diremos que un código $\mathcal{C}$ detecta $d - 1$ errores cuando se da esta situación.

Lema 3.1.7. *Sea $\mathcal{C}$ un código tipo (n, m, d). Entonces, para cualesquiera $x_1, x_2 \in \mathcal{C}$, y para todo $x \in \mathcal{A}^n$, no es posible*

$$d(x_i, v) \leq \left[\frac{d-1}{2}\right], \quad i = 1, 2.$$

Dicho de otro modo,

$$\overline{B}\left(x_1, \left[\frac{d-1}{2}\right]\right) \cap \overline{B}\left(x_2, \left[\frac{d-1}{2}\right]\right) = \emptyset$$

Diremos que el código $\mathcal{C}$ corrige $\left[\frac{d-1}{2}\right]$ errores cuando se da esta situación.

3.2. Códigos lineales

Los códigos lineales se introducen en el caso de que el alfabeto sea un cuerpo finito. A partir de ahora vamos a trabajar con $\mathcal{A} = \mathbb{F}_q$. que denota un cuerpo finito con q elementos. Nótese que si el alfabeto $\mathcal{A}$ es un cuerpo finito entonces $\mathcal{A}^n = \mathbb{F}_q^n$ tiene estructura de espacio vectorial.

Definición 3.2.1. *Un código lineal de longitud n sobre el alfabeto $\mathcal{A} = \mathbb{F}_q$ es un subespacio vectorial de $\mathbb{F}_q^n$.*

Observación 3.2.2. *Un código lineal $\mathcal{C} \subseteq \mathbb{F}_q^n$, como subespacio vectorial que es, tiene una dimensión, digamos $dim(\mathcal{C}) = k$. En estas condiciones es claro que $\mathcal{C}$ es un código tipo (n, q^k, d), con $R(\mathcal{C}) = k/n$ y redundancia $n - k$. Esto es, de las n coordenadas que tiene cada palabra de $\mathcal{C}$, k contienen información, lo cual es perfectamente coherente con el hecho de que $dim(\mathcal{C}) = k$.*

Observación 3.2.3. *Un uso típico de códigos lineales es el siguiente. Supongamos que tenemos un mensaje que queremos transmitir sin errores, escrito como bloques de longitud k sobre un cuerpo finito $\mathbb{F}_q$. Elegimos o diseñamos un código de dimensión k y longitud n, con distancia mínima d adecuada a la tasa de errores del canal de transmisión. supongamos que $\mathcal{C} = \langle u_1, \ldots, u_k \rangle \subset (\mathbb{F}_q)^n$.*

Dado un mensaje $m \in \mathcal{A}^k = \mathbb{F}_q^k$, llamamos la codificación de m al vector del código

$$z = m_1 u_1 + \cdots + m_k u_k \in \mathcal{C}.$$

El vector z es lo que se transmite, y en el canal de transmisión ocurren unos ciertos errores. Podemos denotar la transmisión con errores por

$$z \rightsquigarrow z + e = x,$$

donde e es el vector de errores.

Ahora, supuesto que nuestro código es suficientemente bueno, se procede a la corrección de x, recuperándose z:

$$x \mapsto z,$$

y recuperando z, es posible calcular de nuevo m como las coordenadas de z en la base elegida; esta operación se llama decodificación.
Esquemáticamente,

$$m \mapsto z \rightsquigarrow x \mapsto z \mapsto m.$$

Definición 3.2.4. *Dado $x \in \mathbb{F}_q^n$ definimos su peso como*

$$w(x) = card\{\ i \mid x_i \neq 0\ \} = d(x, 0)$$

Proposición 3.2.5. *Si $\mathcal{C}$ es un código lineal tipo (n, q^k, d), entonces*

$$d = \min_{0 \neq x \in \mathcal{C}} w(x).$$

Proposición 3.2.6 (Cota de Singleton). *Dado un código lineal $\mathcal{C}$ con parámetros (n, q^k, d), se cumple que*

$$d \leq n - k + 1$$

Observación 3.2.7. *La cota de Singleton pone de manifiesto de forma precisa que la distancia mínima y el tamaño del código no pueden crecer a la vez, algo que ya habíamos intuido.*

Definición 3.2.8. *Los códigos que verifican $k+d=n+1$ se llaman códigos de máxima distancia de separación, o códigos MDS.*

Proposición 3.2.9 (**Cota de Plotkin**). *En las condiciones anteriores*

$$d \leq \frac{nq^{k-1}(q-1)}{q^k-1}$$

Definición 3.2.10. *Sea $\mathcal{C} \subset \mathbb{F}_q^n$ un código lineal. Diremos que $\mathcal{M}$ es una matriz generatriz de $\mathcal{C}$ cuando las columnas de $\mathcal{M}$ formen una base de $\mathcal{C}$ como subespacio vectorial.*

Definición 3.2.11. *Sea $\mathcal{C} \subset \mathbb{F}_q^n$ un código lineal. Si $\mathcal{C}$ viene expresado por un sistema de ecuaciones implícitas independientes,*

$$AX = 0 \in \mathscr{M}((n-k)\times 1, \mathbb{F}_q)$$

diremos que A es una matriz de control o de paridad de $\mathcal{C}$.

Proposición 3.2.12. *Dado un vector $x \in \mathbb{F}_q^n$,*

$$x \in \mathcal{C} \iff Ax = 0 \in \mathscr{M}((n-k)\times 1, \mathbb{F}_q) \iff rg(\mathcal{M}) = rg(\mathcal{M}|x).$$

Proposición 3.2.13. *La distancia mínima de $\mathcal{C}$ es la menor cantidad de columnas de A que necesitamos para formar un conjunto linealmente independiente.*

Definición 3.2.14. *Llamamos síndrome de v al vector*

$$s(v) = Av \in \mathbb{F}_q^{n-k}$$

Proposición 3.2.15. *Algunas propiedades del síndrome son:*

1. $s(z) = 0 \iff z \in \mathcal{C}$

2. *Si* $v = x + e$, *y* $x \in \mathcal{C}$, *entonces* $s(v) = s(e)$.

3. *El síndrome de un vector es una combinación lineal de las columnas de* A *correspondientes a las coordenadas en las que se ha cometido un error de transmisión.*

Observación 3.2.16. *Dos palabras tienen el mismo síndrome si y sólo si su diferencia es una palabra del código, es decir,*

$$s(u) - s(v) = s(u - v) = 0 \iff u - v \in \mathcal{C}$$

y esto permite interpretar el síndrome en términos de espacios cocientes.

Dado un código lineal $\mathcal{C} \subset \mathbb{F}_q^n$ *de tamaño* q^k, *consideramos el espacio* $\mathbb{F}_q^n/\mathcal{C}$, *que es un espacio vectorial de dimensión* $n - k$. *Sus elementos son las clases*

$$x + \mathcal{C} = \{x + u \mid u \in \mathcal{C}\}.$$

Cada clase tiene q^k *elementos. Dos vectores* u, v *están en la misma clase si y sólo si sus síndromes coinciden:*

$$u - v \in \mathcal{C} \iff u + \mathcal{C} = v + \mathcal{C} \iff s(u) = s(v).$$

Si ahora pensamos en un vector e *como en un vector de errores en la transmisión, la clase de* $e + \mathcal{C}$ *consiste en todas las palabras del código afectadas del mismo error, y todas de síndrome igual a* $s(e)$. *Si además supiéramos que no hay dos errores con el mismo síndrome, podríamos recuperar la palabra original simplemente restando* e.
Para ello, necesitamos ver algunas condiciones de unicidad.

Definición 3.2.17. *Diremos que un elemento* u *es el líder de una clase* $\mathbb{F}_q^n/\mathcal{C}$ *cuando sea el único elemento de peso mínimo.*

Proposición 3.2.18. *Sea* $\mathcal{C} \subset \mathbb{F}_q^n$ *un código lineal de tipo* (n, q^k, d). *Sea* $e \in \mathbb{F}_q^n$ *tal que* $w(e) \leq [\frac{d-1}{2}]$. *Entonces, para todo* $u \in e + \mathcal{C}$ *distinto de* e, *se tiene que* $w(u) > [\frac{d-1}{2}]$. *En particular,* e *es el líder de su clase.*

La proposición anterior permite usar un algoritmo decodificador llamado algoritmo del líder o algoritmo de síndromes y errores.

4

CÓDIGOS DE GOPPA

4.1. Definición y propiedades

Definición 4.1.1. *[3] Sean $L = (a_1, \ldots, a_n) \in \mathbb{F}_{q^m}^n$ una n-tupla de n elementos distintos de $\mathbb{F}_{q^m}$ y $g(x) \in \mathbb{F}_{q^m}[x]$ un polinomio de grado t tal que $g(a_j) \neq 0$ $\forall j = 1, \ldots, n$. Se define el código de Goppa respecto de L y g como:*

$$\Gamma(L, g) := \left\{ c \in \mathbb{F}_q^n \quad \middle| \quad \sum_{i=1}^{n} \frac{c_i}{x - a_i} \equiv 0 \mod g(x) \right\},$$

que se trata de un código lineal definido en $\mathbb{F}_q$ de longitud n.

Llamaremos al polinomio $g(x)$ Polinomio de Goppa y denotaremos el código de Goppa $\Gamma(L, g(x))$.

Observación 4.1.2. *[4] El término $\frac{1}{x-a_i}$ se puede ver como un polinomio $p_i(x)$ módulo $g(x)$*

$$\frac{1}{x - a_i} \equiv p_i(x) = p_{i1} + p_{i2}x + \cdots + p_{it}x^{t-1}$$

Esto de debe a la forma que tienen los elementos en el cuerpo $\mathbb{F}_{q^m} g(x)$. Por lo tanto podemos reescribir la ecuación de la definición 4.1.1 como

$$\sum_{i=1}^{n} c_i p_i(x) \equiv 0 \mod g(x). \tag{4.1}$$

Para la codificación y decodificación, necesitamos una matriz de chequeo de paridad H del código. Para ello vamos a enunciar el siguiente teorema.

Teorema 4.1.3. *[5] Sea un código de Goppa* $\Gamma(L,g) = \{c \in \mathbb{F}_q^n \ : \ cH'^t = 0\}$, *donde* $g(x)$ *tiene grado* t, $L = (a_1, \ldots, a_n)$ *y definimos H' como*

$$H' = \begin{pmatrix} a_1^{t-1}g(a_1)^{-1} & a_2^{t-1}g(a_2)^{-1} & \cdots & a_n^{t-1}g(a_n)^{-1} \\ a_1^{t-2}g(a_1)^{-1} & a_2^{t-2}g(a_2)^{-1} & \cdots & a_n^{t-2}g(a_n)^{-1} \\ \vdots & \vdots & \ddots & \vdots \\ a_1 g(a_1)^{-1} & a_2 g(a_2)^{-1} & \cdots & a_n g(a_n)^{-1} \\ g(a_1)^{-1} & g(a_2)^{-1} & \cdots & g(a_n)^{-1} \end{pmatrix} \tag{4.2}$$

entonces H' *es una matriz de control.*

Demostración.

Para demostrar que H' es una matriz de control del Código de Goppa, debemos recordar que si $c \in \Gamma(L,g)$ si y sólo si se cumple (4.1) donde podemos escribir $\frac{1}{x-a_i} \equiv p_1(x) = p_{i1} + p_{i2}x + \cdots + p_{it}x^{t-1}$ *mod* $g(x)$. Una de las posibles matrices de control debe cumplir que $cH^t = 0$, por lo que tenemos que

$$H = \begin{pmatrix} p_{11} & \cdots & p_{n1} \\ \vdots & \ddots & \vdots \\ p_{1t} & \cdots & p_{nt} \end{pmatrix} \tag{4.3}$$

Verificando que $cH' = 0$, tenemos que

$$(c_1, \cdots, c_n) \times \begin{pmatrix} p_{11} & \cdots & p_{n1} \\ \vdots & \ddots & \vdots \\ p_{1t} & \cdots & p_{nt} \end{pmatrix}' = \left(\sum_{i=1}^{n} c_i p_{i1}, \cdots, \sum_{i=1}^{n} c_i p_{it} \right) \tag{4.4}$$

Podemos ver que la ecuación anterior es cero módulo $g(x)$ por definición. Luego para poder determinar los factores p_{ij} de la matriz H debemos reescribir el término $\frac{1}{x-a_i}$ de la siguiente manera. Primero definamos $u(x)$

$$u(x) = -g(x)g(a_i)^{-1} + 1 = -(g(c) - g(a_i)))g(a_i)^{-1}$$

Podemos ver que $u(x) \equiv\ 1\ mod\ g(x)$, entonces

$$\frac{1}{x - a_i} \equiv \frac{u(x)}{x - a_i} \equiv -\frac{g(x) - g(a_i)}{x - a_i} g(a_i)^{-1} \equiv p_i(x).$$

Vamos a definir ahora $h_i = g(a_i)^{-1}$ y recordemos que
$g(x) = g_0 + g_1 x + \cdots + g_t x^t$. Reemplazando en la ecuación anterior $g(x)$, encontramos

$$p_i = -\frac{g_t \times (x^t - a_i^t) \times \cdots \times g_1 \times (x - a_i)}{x - a_i} \times h_i \tag{4.5}$$

Esta fracción puede ser escrita como

$$g_t(x^{t-1} + x^{t-2}a_i + \cdots + a_i^{t-1}) + g_{t-1}(x^{t-2} + x^{t-3}a_i + \cdots + a_i^{t-2}) + \cdots + g_2(x + a_i) + g_1$$

Si sustituimos $p_i(x) = p_{i1} + p_{i2}x + \cdots + p_{it}x^{t-1}$, podemos encontrar la siguiente expresión para los p_{ij}

$$\begin{cases} p_{i1} & = -(g_t a_i^{t-1} + g_{t-1}a_i^{t-2} + \cdots + g_2 a_i + g_1)h_i \\ p_{i2} & = \quad -(g_t a_i^{t-2} + g_{t-1}a_i^{t-3} + \cdots + g_2)h_i \\ & \vdots \\ p_{i(t-1)} & = \quad -(g_t a_i + g_{t-1})h_i \\ p_{it} & = \quad -(g_t)h_i \end{cases} \tag{4.6}$$

Combinando (4.3) y (4.6), encontramos que $H = ABD$ para

$$A = \begin{pmatrix} -g_t & -g_{t-1} & -g_{t-2} & \cdots & -g_1 \\ 0 & -g_t & -g_{t-1} & \cdots & -g_2 \\ 0 & 0 & -g_t & \cdots & -g_3 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & -g_t \end{pmatrix}. \tag{4.7}$$

$$B = \begin{pmatrix} a_1^{t-1} & a_2^{t-1} & \cdots & a_n^{t-1} \\ a_1^{t-2} & a_2^{t-2} & \cdots & a_n^{t-2} \\ \vdots & \vdots & \ddots & \vdots \\ a_1 & a_2 & \cdots & a_n \\ 1 & 1 & \cdots & 1 \end{pmatrix}. \tag{4.8}$$

$$D = \begin{pmatrix} h_1 & 0 & 0 & \cdots & 0 \\ 0 & h_2 & 0 & \cdots & 0 \\ 0 & 0 & h_3 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & h_n \end{pmatrix}. \tag{4.9}$$

Por tanto, $c \in \Gamma(L, g)$ si y solo si $cH^t = 0$, lo que implica $c(ABD)^t = c(D^t B^t A^t) = 0$, lo cual nos da que $c(D^t B^t) = c(BD)^t = 0$, debido a que la matriz A es invertible, al ser el $det(A) = g_t^t \neq 0$, ya que si g_t fuera 0 nuestro polinomio g(x) tendría grado t - 1 y por definición sabemos que tiene grado t. Entonces $H' = (BD)$. □

Proposición 4.1.4. *Una vez se ha calculado la matriz H', se puede obtener G a partir de la igualdad $GH'^t = 0$. La filas de G forman una base del código como subespacio vectorial. La matriz G nos permite codificar y decodificar.*

Proposición 4.1.5. *Sea* $g(x) = g_0 + g_1x + \cdots + g_tx^t \in \mathbb{F}_{q^m}[x]$ *un polinomio de Goppa de grado* t *y* $L = (a_1, \ldots, a_n) \in F_{q^m}^n$. *El código de Goppa* $\Gamma(L, g)$ *tiene parámetros* (n, k, d) *donde*

$$k \geq n - mt$$
$$d \geq t + 1$$

Puede corregir $\left[\frac{t}{2}\right]$ *errores.*

Proposición 4.1.6. *Si el código de Goppa* $\Gamma(L, g)$ *se define sobre* $\mathbb{F}_2$, *con* $L = (a_1, \ldots, a_n) \in \mathbb{F}_{2^m}^n$ *y el polinomio de Goppa* $g(x) \in \mathbb{F}_{2^m}[x]$ *de grado* t *es separable, es decir, el polinomio tiene todas sus raíces simples, entonces la distancia mínima del código* $d \geq 2t + 1$.
Por tanto, para estos casos doblamos la capacidad correctora de los códigos de Goppa. Corrige [t] errores.

4.2. Codificación

La codificación de los códigos de Goppa supone, dividir el mensaje en bloques de tamaño k y luego multiplicar cada bloque por la matriz generatriz G. El vector resultante forma el conjunto de palabras códigos, es decir:

$$(m_1, \ldots, m_k) \times G = (c_1, \ldots, c_n).$$

4.3. Decodificación

Consideramos el Código de Goppa definido sobre el cuerpo $\mathbb{F}_2$ [6]. Sea una palabra del código $c = (c_1, \ldots, c_n) \in \Gamma(L, g)$, que enviamos a un receptor, el cual recibe $y = (y_1, ..., y_n) = c + e$ donde $e = (e_1, \ldots, e_n)$ es el vector de error, donde suponemos $w(e) \leq t$, es decir, que el peso del error no supera el número de errores que puede corregir el código.

Introducimos los siguientes conceptos para la corrección de los errores producidos.

Definición 4.3.1. *Llamamos síndrome del vector* $y = (y_1, \ldots, y_n)$ *y lo denotamos* $s(x)$ *al valor:*

$$s(x) = \sum_{i=1}^{n} \frac{y_i}{x - a_i} \mod g(x) = \sum_{i=1}^{n} \frac{e_i}{x - a_i} \mod g(x)$$

Definición 4.3.2. *Llamamos polinomio localizador de errores al polinomio de la forma*

$$L(x) = \prod_{i|e_i \neq 0} (x - a_i)$$

Definición 4.3.3. *Llamamos polinomio evaluador de errores al polinomio de la forma*

$$E(x) = \sum_{i|e_i \neq 0} e_i \prod_{j|e_j \neq 0, j \neq i} (x - a_j)$$

Vamos a detallar el Algoritmo de Patterson,para la corrección de errores, cuyo uso es exclusivo para Códigos de Goppa binarios.

4.4. Algoritmo de Patterson

Sea el código de Goppa binario, $\Gamma(L, g)$, Su polinomio $g(x) = g_0 + g_1 x + \cdots + g_t x^t$ tiene coeficientes en $\mathbb{F}_{2^m}$ y es de grado t y $L = (a_1, \ldots, a_n) \in \mathbb{F}_{2^m}^n$.

Con el algoritmo calculamos primero el síndrome del vector y una vez calculado este, obtenemos el polinomio localizador de errores L(x):

1. Buscamos $f(x)$ tal que que $s(x)f(x) \equiv 1 \mod g(x)$. Si $f(x) = x$ tendríamos que $L(x) = x$ y acabamos. Si no, seguimos los pasos siguientes.

2. Buscamos el polinomio $h(x)$ que cumpla $h^2(x) \equiv f(x) + x \mod g(x)$

3. Calculamos los polinomios $a(x)$ y $b(x)$ de grado mínimo que cumplan que $h(x)b(x) \equiv a(x) \mod g(x)$

4. Construimos el polinomio localizador de errores $L(x) = a^2(x) + xb^2(x)$

Calculando las raíces del polinomio L(x), las cuales deben ser elementos del vector L, obtenemos las posiciones en las que hay errores, $(i|e_i \neq 0)$. Las posiciones de errores se corresponden con las posiciones que ocupan en el vector L los elementos que son raíces del polinomio localizador de errores, L(x).
La palabra codificada sin errores se obtiene restando el vector de errores $e = (e_1, ..., e_n)$:

$$c = y - e = y + e$$

5

CRIPTOSISTEMA DE MCELIECE

5.1. Introducción

El criptosistema McEliece es un criptosistema de clave pública el cual fue desarrollado por Robert McEliece en 1978 [6].

Se basa en la teoría de códigos y su seguridad se encuentra principalmente en que la matriz generatriz del código parece aleatoria y en la dificultad de decodificación de un código lineal del que no conocemos su estructura, al tratarse este de un problema computacionalmente difícil (NP-Hard) cuando el tamaño del código es grande.

Para el criptosistema de McEliece se emplean códigos de Goppa binarios debido a que son seguros, sencillos de construir y emplear y presentan mejores propiedades que los códigos de Goppa construidos sobre otros cuerpos finitos.

5.2. Descripción

Para construir el Criptosistema de McEliece, necesitamos construir primero un Código de Goppa binario, es decir, escogemos un polinomio separable, para que el número de errores que se pueda corregir sea mayor, con coeficientes en $\mathbb{F}_{2^m}$ de grado t y un vector $L \in \mathbb{F}_{2^m}^n$. A partir de estos elementos calculamos la matriz de paridad H y la matriz generatriz del código, $G \in \mathscr{M}(k \times n, \mathbb{F}_2)$. Para estos códigos hemos visto el algoritmo de Patterson, a partir del cual podemos realizar la decodificación de hasta t errores de forma eficiente.

Para calcular la matriz G' necesitamos las matrices:

- Matriz generatriz, G, del Código de Goppa
- S, es una matriz no singular cuadrada de tamaño k
- Matriz P, binaria, cuadrada de tamaño n, donde sus elementos son nulos salvo uno por fila y columna.

Una vez obtenidas estas tres matrices $G' = S * G * P$

$S * G$ es también una matriz generatriz del Código de Goppa creado y G' es una matriz generatriz de un código con los mismos parámetros que el generado por G, pero que esconde a la perfección el la estructura del Código de Goppa.

Entonces las claves pública, κ_p, y privada, κ_s, del criptosistema de McEliece son respectivamente:

$$\kappa_p = (G', t) \quad y \quad \kappa_s = (G, S, P, D)$$

donde D es en esta ocasión el algoritmo de decodificación de Patterson.

5.3. Cifrado y descifrado

Los pasos para una comunicación con este criptosistema son los siguientes:

- Para cifrar un mensaje de tamaño k, $m = (m_1, \ldots, m_k)$, destinado a un usuario con clave pública $\kappa_p = (G', t)$, tenemos que multiplicar el mensaje m por la matriz G' y obtenemos el vector c de tamaño n

  $$c = mG' = (c_1, \ldots, c_n)$$

  Ahora añadimos una cantidad de errores, de tamaño menor o igual a t, al vector c, es decir, tomamos $e = (e_1, \ldots, e_n)$ con $w(e) \leq t$ y sumamos vectores, $y = c + e$.

  $$y = mG' + e = m(SGP) + e$$

- Para realizar el descifrado del mensaje, primero se multiplica y por la inversa de la matriz de permutación, P^{-1},

  $$y' = yP^{-1} = mSG + eP^{-1}$$

  Ahora empleamos el algoritmo de Patterson para corregir los errores. Este lo podemos usar ya que la matriz SG es también una matriz generatriz del código de Goppa a y que tanto el vector e como el vector eP^{-1} tienen el mismo peso.

  Una vez obtenido $c' = mSG$ resolvemos el sistema de ecuaciones y tenemos $m' = mS$.

  Para finalizar multiplicamos por la inversa de S y recuperamos el mensaje original $m = m'S^{-1}$

6

IMPLEMENTACIÓN EN SAGEMATH

6.1. Construcción del Código de Goppa

Construimos los elementos que componen el Código de Goppa [7] [8] [9].

```
# Definimos el cuerpo donde vamos a trabajar
m = 4;rango = 3;N = 2^m
K_.<a> = GF(2)
F.<a> = GF(2^m)
# Creamos el anillo de polinomios
PR = PolynomialRing(F,'X')
X = PR.gen()
g = X^3+X+1                    # Polinomio de Goppa
L = [a^i for i in range(N)]    # Soporte del codigo
print("Polinomio de Goppa","=",g)
print("N-tupla","=",L)
```

$$g = X^3 + X + 1$$

$$L = [1, a, a^2, a^3, a+1, a^2+a, a^3+a^2, a^3+a+1, a^2+1, a^3+a,$$
$$a^2+a+1, a^3+a^2+a, a^3+a^2+a+1, a^3+a^2+1, a^3+1, 1]$$

La matriz de control del Código de Goppa la obtenemos mediante la multiplicación de las siguientes matrices:

```
A = matrix(F,rango,rango)
for i in range(rango):
    count = rango - i
    for j in range(rango):
        if i > j:
            A[i,j]=g.list()[count]
            count = count + 1
        if i < j:
            A[i,j] = 0
        if i == j:
            A[i,j] = 1

print ("Matriz A: ")
show(A)
```

$$A = \begin{pmatrix} 1\,0\,0 \\ 0\,1\,0 \\ 1\,0\,1 \end{pmatrix}$$

```
B = matrix([[L[j]^i for j in range(N)] for i in range(rango)])
print ("Matriz B: ")
show(B)
```

$$B = \left(\begin{array}{ccccccccc} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & \cdots \\ 1 & a & a^2 & a^3 & a+1 & a^2+a & a^3+a^2 & a^3+a+1 & \cdots \\ 1 & a^2 & a+1 & a^3+a^2 & a^2+1 & a^2+a+1 & a^3+a^2+a+1 & a^3+1 & \cdots \end{array} \right.$$

$$\left. \begin{array}{ccccccccc} \cdots & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ \cdots & a^2+1 & a^3+a & a^2+a+1 & a^3+a^2+a & a^3+a^2+a+1 & a^3+a^2+1 & a^3+1 & 1 \\ \cdots & a & a^3 & a^2+a & a^3+a+1 & a^3+a & a^3+a^2+a & a^3+a^2+1 & 1 \end{array} \right)$$

```
D = diagonal_matrix([1/g(L[i]) for i in range(N)])
print ("Matriz D: ")
show(D)
```

$$D = \left(\begin{array}{ccccccccc} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots \\ 0 & a^2+1 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots \\ 0 & 0 & a & 0 & 0 & 0 & 0 & 0 & \cdots \\ 0 & 0 & 0 & a^3+a^2+a & 0 & 0 & 0 & 0 & \cdots \\ 0 & 0 & 0 & 0 & a^2 & 0 & 0 & 0 & \cdots \\ 0 & 0 & 0 & 0 & 0 & a^2+a+1 & 0 & 0 & \cdots \\ 0 & 0 & 0 & 0 & 0 & 0 & a^3+a+1 & 0 & \cdots \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & a^2+a+1 & \cdots \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots \end{array} \right.$$

$$
\begin{array}{ccccccccc}
\cdots & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\cdots & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\cdots & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\cdots & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\cdots & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\cdots & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\cdots & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\cdots & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\cdots & a+1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\cdots & 0 & a^3+a^2+1 & 0 & 0 & 0 & 0 & 0 & 0 \\
\cdots & 0 & 0 & a^2+a & 0 & 0 & 0 & 0 & 0 \\
\cdots & 0 & 0 & 0 & a^2+a & 0 & 0 & 0 & 0 \\
\cdots & 0 & 0 & 0 & 0 & a^3+1 & 0 & 0 & 0 \\
\cdots & 0 & 0 & 0 & 0 & 0 & a^2+a+1 & 0 & 0 \\
\cdots & 0 & 0 & 0 & 0 & 0 & 0 & a^2+a & 0 \\
\cdots & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{array}\right)
$$

La matriz de control es el resultado de la multiplicación de estas tres matrices anteriores.

```
H = A*B*D
print("H=")
show(H)
```

$$
H = \left(\begin{array}{ccccccccc}
1 & a^2+1 & a & a^3+a^2+a & a^2 & a^2+a+1 & a^3+a+1 & a^2+a+1 & \cdots \\
1 & a^3+a & a^3 & a^3+1 & a^3+a^2 & 1 & a^3+a^2+1 & a^2 & \cdots \\
0 & a & a^2 & a^3+a & a+1 & 1 & a^3 & a^3+a^2+1 & \cdots
\end{array}\right.
$$

$$
\left.\begin{array}{ccccccccc}
\cdots & a+1 & a^3+a^2+1 & a^2+a & a^2+a & a^3+1 & a^2+a+1 & a^2+a & 1 \\
\cdots & a^3+a^2+a+1 & a^3+a+1 & 1 & a & a^3+a^2+a & a^2+1 & a+1 & 1 \\
\cdots & a^2+1 & a^3+a^2+a+1 & 1 & a^3+1 & a^3+a^2 & a^3+a+1 & a^3+a^2+a & 0
\end{array}\right)
$$

Las entradas de la matriz H son elementos de $\mathbb{F}_{2^4}$. Vamos a expresar cada uno de estos elementos como un vector de $\mathbb{F}_2^4$. De esta forma obtenemos la matriz de control de un codigo lineal con la misma longitud que el anterior pero conmayor distancia mínima.

```
H_Goppa_K = matrix(K_, m*H.nrows(),H.ncols())
for i in range(H.nrows()):
    for j in range(H.ncols()):
        be = bin(eval(H[i,j]._int_repr()))[2:];
        be = '0'*(m-len(be))+be; be = list(be);
        H_Goppa_K[m*i:m*(i+1),j]=vector(map(int,be));
show(H_Goppa_K)
```

$$H\_Goppa\_k = \begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 0 \end{pmatrix}$$

A partir de la matriz de control obtenemos la patriz generatriz del código de Goppa.

```
Krnl = H_Goppa_K.right_kernel();
G = Krnl.basis_matrix();
print("G=")
show(G)
```

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

El Código de Goppa que hemos creado es un código lineal que tiene a G como matriz generatriz.

```
C=LinearCode(G) #codigo lineal generado por la matriz G
C
```

6.1.1. Ejemplo de codificación y decodificación

Vamos a realizar un ejemplo de codificación y decodificación del código creado anteriormente. Para decodificar usamos el Algoritmo de Patterson el cual construimos con ayuda de las siguientes funciones auxiliares.

```
# --------------------
# Funcion auxiliar que permite descomponer un polinomio en irreducibles
# --------------------
def descomponer_polinomio(p):
    # El siguiente metodo permite descomponer
    # un polinomio p en factores irreducibles p(z) = p0 (z) + z p1 (z)
    # Entrada: Polinomio p
    Phi1 = p.parent()
    p0 = Phi1([sqrt(c) for c in p.list()[0::2]])
    p1 = Phi1([sqrt(c) for c in p.list()[1::2]])
    return (p0,p1)
```

Algoritmo de euclides extendido: Obtener MCD y los s,t que lo generan.

```
def algoritmo_euclides_extendido(self, other):
    delta = self.degree() #grado de polinomio 1
    if other.is_zero(): # si el polinomio introducido es
        ring = self.parent() #comprobamos el cuerpo en el que trabajamos
        return self, R.one(), R.zero() #mcd = mismo polinomio y devuelve
                # un uno (s) y un cero (t) en el cuerpo que trabajamos.
    # mcd (a,b) = as+bt

    ring = self.parent() #comprobamos el cuerpo en el que trabajamos
    a = self # guardamos una copia del primer polinomio 1 (self)
    b = other # guardamos una copia del segundo polinomio (other)

    s = ring.one() # guardamos en s el uno del anillo
    t = ring.zero() # guardamos en t el cero del anillo

    resto0 = a
    resto1 = b

    while true:
        cociente,resto_auxiliar = resto0.quo_rem(resto1)
        resto0 = resto1             # La funcion quo_rem de Sage
        resto1 = resto_auxiliar     # devuelve el cociente y el resto.

        s = t
        t = s - t*cociente

        if resto1.degree() <= floor((delta-1)/2)
            and resto0.degree() <= floor((delta)/2):
                break

    V = (resto0-a*s)//b
    coeficiente_lider = resto0.leading_coefficient()
     # guardamos el coeficiente lider del resto 0

    return resto0/coeficiente_lider, s/coeficiente_lider, V/coeficiente_lider
```

Función que calcula la inversa de un polinomio.

```
def inversa_g(p,g):
    # Input: Polinomios p y g
    # Output:retornar polinomio P modulo g
    (d,u,v) = xgcd(p,g)
    return u.mod(g)
```

Función del algoritmo de decodificación de Patterson.

```
def decodePatterson(y):
    # Calculamos primero el vector alpha con los elementos primitivos.
    alpha = vector(H*y)

    # Consideramos nuestras matrices T,Y,Z
    # definidas así como nuestro polinomio irreducible g
    polinomioS = PR(0) # Inicializa como el polinomio 0 del anillo
    for i in range(len(alpha)):
        polinomioS = polinomioS + alpha[i]*(X^(len(alpha)-i-1))
        # Lo vamos rellenando con los alpha
    vector_g = descomponer_polinomio(g)
    w = ((vector_g[0])*inversa_g(vector_g[1],g)).mod(g)
    vector_t = descomponer_polinomio(inversa_g(polinomioS,g) + X)

    R = (vector_t[0]+(w)*(vector_t[1])).mod(g)

    (A,aux,B) = algoritmo_euclides_extendido(g,R)

    # Definimos el polinomio sigma
    sigma = A^2+X*(B^2)

    # Vamos comprobando uno a uno los coeficientes de sigma
    # para asi determinar el conjunto de
    #posiciones de error E - {i tal que e_i es distinto de 0}
    for i in range(N):
        if (sigma(a^i)==0): # an error occured
            print ("Error encontrado en la posición: " + str(i))
            y[i] = y[i] + 1
    return y
```

Ejemplo de Codificación. En él calculamos un vector aleatoria u, con tamaño igual a dimensión del código, lo codificamos multiplicando dicho vector por la matriz G. por último creamos un vector aleatorio con dos errores y lo introducimos los errores en el vector codificado.

```
u = vector(K_,[randint(0,1) for _ in range(G.nrows())])
c = u*G
print ('Vector u')
show(u)
print ('Vector c')
show(c)
e = vector(K_,N)
e[8] = 1
e[9] = 1
print ('Vector de errores e')
show(e)
y = c + e
print ("Vector codificado y")
show(y)
```

$$u = (0,\ 0,\ 1,\ 1)$$

$$c = (0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0)$$

$$e = (0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0)$$

$$y = (0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0)$$

Decodificación y mensaje recibido

```
sol = decodePatterson(y)
print("sol=")
show(sol)
mensaje = (G.transpose()\sol)
print("mensaje")
show(mensaje)
```

Error encontrado en la posición: 8
Error encontrado en la posición: 9

$$Sol = (0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0)$$

$$mensaje = (0, 0, 1, 1)$$

6.2. Criptosistema de McEliece

Calculamos la matriz del Criptosistema de McEliece usando la matriz generatriz del Código de Goppa calculado y las siguientes matrices.

Matriz aleatoria invertible de orden kxk

```
S = random_matrix(GF(2), N-m*rango) # matriz binaria no singular.

while (S.determinant()==0):
    S = random_matrix(GF(2), N-m*rango)
print("S=")
show(S)
```

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

Matriz de permutación de orden NxN

```
rng = range(N)

P = matrix(GF(2),N); #matriz de permutacion.

for i in range(N):
    p = floor(len(rng)*random());
    P[i,rng[p]]=1;
    rng=list(rng[:p])+list(rng[p+1:]);
print("P=")
show(P)
```

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

Definimos la matriz del Criptosistema de McEliece

```
G_gorro = S*G*P
print("G_gorro")
show(G_gorro)
```

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

6.2.1. Ejemplo sencillo del Criptosistema de McEliece

Vamos a tomar como mensaje un vector aleatorio u del longitud la dimensión del código.

```
u = vector(K_,[randint(0,1) for _ in range(G_gorro.nrows())])
c = u*G_gorro
print ('Vector u');
show(u)
print ('Vector c');
show(c)
e = vector(K_,N)
e[8] = 1
e[9] = 1
print ('Vector de errores e') ; show(e)
y = c + e
print ("Vector codificado y") ; show(y)
```

$$u = (1,\ 0,\ 0,\ 1)$$

$$c = (1,\ 0,\ 1,\ 0,\ 1,\ 1,\ 0,\ 0,\ 0,\ 0,\ 1,\ 0,\ 1,\ 1,\ 0,\ 0)$$

$$e = (0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 1,\ 1,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0)$$

$$y = (1,\ 0,\ 1,\ 0,\ 1,\ 1,\ 0,\ 0,\ 1,\ 1,\ 1,\ 0,\ 1,\ 1,\ 0,\ 0)$$

Decodificación:

```
yP = y*(P.inverse())
yd = decodePatterson(yP)
print("yd=")
show(yd)
corregido = (G.transpose()\yd)*S.inverse()
print("corregido=")
show(corregido)
```

Error encontrado en la posición: 3
Error encontrado en la posición: 12

$$yd = (0,\ 1,\ 1,\ 0,\ 1,\ 1,\ 0,\ 0,\ 1,\ 0,\ 1,\ 0,\ 0,\ 0,\ 0,\ 1)$$
$$corregido = (1,\ 0,\ 0,\ 1)$$

6.2.2. Ejemplo complejo del Criptosistema de McEliece

En esta ocasión vamos a codificar y decodificar cualquier palabra, número o carácter.

Para ello primero tenemos que importar las funciones que convierten caracteres en números binarios y al contrario.

De caracteres a binarios:

```
from sage.crypto.util import ascii_to_bin
m=ascii_to_bin("MR")
m
```

$$m = 0100110101010010$$

Función inversa a la anterior:

```
from sage.crypto.util import bin_to_ascii
h=bin_to_ascii("0100110101010010")
h
```

$$h = \texttt{MR}$$

Como necesitamos vectores, creamos una función que introducimos una palabra, esta se convierte en números binarios y lo muestra en forma de vector.

```
def DeTextoAVectorBin(mensaje):
    texto=ascii_to_bin(mensaje)
    lista=list(texto)
    vectori =vector([K_(str(list(lista)[i])) for i in range(len(lista))])
                        #de cadena a  lista y de lista a vector
    return vectori

Iniciales=DeTextoAVectorBin("MR")
Iniciales
```

$$Iniciales = (0,\ 1,\ 0,\ 0,\ 1,\ 1,\ 0,\ 1,\ 0,\ 1,\ 0,\ 1,\ 0,\ 0,\ 1,\ 0)$$

Necesitamos que la cantidad de números binarios resultante de la transformación de los caracteres sea múltiplo de la dimensión del Código de Goppa, por tanto si no es así rellenamos con ceros al final.

```
def hacerconmultiplo2(mn):
    z=mod(len(mn),C.dimension())
    if z == 0:
        return mn
    else:
        return(hacerconmultiplo2(mn+[0]))

multi=hacerconmultiplo2([1,0,1,1,1])
multi
```

$$multi = [1, 0, 1, 1, 1, 0, 0, 0]$$

Por último tenemos que dividir la cantidad de números binarios en vectores de longitud la dimensión del código, para ello hemos creado la siguiente función:

```
def dividirlista2(vectorbin):
    iterar=len(vectorbin)/C.dimension();
    bb=[vectorbin[i*C.dimension():(i+1)*C.dimension()] for i in range(floor(iterar))];
    return(bb)

divid=dividirlista2((0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0))
divid
```

$$divid = [(0,1,0,0)\,,(1,1,0,1)\,,(0,1,0,1)\,,(0,0,1,0)]$$

Por último unimos estas tres funciones anteriores para crear otro que introduciendo cualquier palabra o número devuelva su conversión a números binarios en una lista con vectores de la longitud de la dimensión del Código de Goppa.

```
def DeTextoAVectorBin23(mensaje):     #esta es la funcion buena
    texto=ascii_to_bin(mensaje)
    lista=list(texto)
    lista=hacerconmultiplo2(lista)
    comoNecesito =vector([K_(str(list(lista)[i])) for i in range(len(lista))])
    dividido elvector =dividirlista2(comoNecesito)
    return dividido elvector

MR=DeTextoAVectorBin23("MR")
MR
```

$$MR = [(0,\ 1,\ 0,\ 0)\,,(1,\ 1,\ 0,\ 1)\,,(0,\ 1,\ 0,\ 1)\,,(0,\ 0,\ 1,\ 0)]$$

Con todo ello definido, codificamos y decodificamos las siglas TFG:

```
u = DeTextoAVectorBin23("TFG")
c = [u[i]*G_gorro for i in range(len(u))]
print ('Vector u')
show(u)
print ('Vector c')
show(c)
e=[];
while len(e) <len(u):
    qw=random_vector(K_,N);
    while qw.hamming_weight()!=C.minimum_distance():
        qw=random_vector(K_,N)  #definir vectores aleatorios
    e=e+[qw]                    #de errores con el peso adecuado
print ('Vector de errores e')
show(e)
y = [c[i] + e[i] for i in range(len(c))]
print ("Vector codificado y")
show(y)
```

$$u = [(0,\ 1,\ 0,\ 1)\,,(0,\ 1,\ 0,\ 0)\,,(0,\ 1,\ 0,\ 0)\,,(0,\ 1,\ 1,\ 0)\,,(0,\ 1,\ 0,\ 0)\,,(0,\ 1,\ 1,\ 1)]$$

$$\begin{aligned} c = [&(1,\ 1,\ 0,\ 1,\ 0,\ 1,\ 0,\ 0,\ 1,\ 1,\ 0,\ 0,\ 1,\ 1,\ 1,\ 0)\,,(0,\ 0,\ 1,\ 1,\ 1,\ 0,\ 1,\ 1,\ 0,\ 0,\ 1,\ 1,\ 0,\ 1,\ 0,\ 1)\,,\\ &(0,\ 0,\ 1,\ 1,\ 1,\ 0,\ 1,\ 1,\ 0,\ 0,\ 1,\ 1,\ 0,\ 1,\ 0,\ 1)\,,(0,\ 0,\ 1,\ 0,\ 1,\ 0,\ 1,\ 1,\ 0,\ 0,\ 1,\ 1,\ 0,\ 0,\ 0,\ 1)\,,\\ &(0,\ 0,\ 1,\ 1,\ 1,\ 0,\ 1,\ 1,\ 0,\ 0,\ 1,\ 1,\ 0,\ 1,\ 0,\ 1)\,,(1,\ 1,\ 0,\ 0,\ 0,\ 1,\ 0,\ 0,\ 1,\ 1,\ 0,\ 0,\ 1,\ 0,\ 1,\ 0)] \end{aligned}$$

$$e = [(0,\ 1,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 1,\ 0,\ 0,\ 0)\,, (0,\ 0,\ 0,\ 0,\ 1,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 1,\ 0,\ 0,\ 0,\ 0)\,,$$
$$(0,\ 0,\ 0,\ 0,\ 1,\ 0,\ 1,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0)\,, (0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 1,\ 1,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0)\,,$$
$$(1,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 1,\ 0,\ 0)\,, (0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 1,\ 0,\ 0,\ 0,\ 1,\ 0,\ 0,\ 0,\ 0)]$$

$$y = [(1,\ 0,\ 0,\ 1,\ 0,\ 1,\ 0,\ 0,\ 1,\ 1,\ 0,\ 0,\ 0,\ 1,\ 1,\ 0)\,, (0,\ 0,\ 1,\ 1,\ 0,\ 0,\ 1,\ 1,\ 0,\ 0,\ 1,\ 0,\ 0,\ 1,\ 0,\ 1)\,,$$
$$(0,\ 0,\ 1,\ 1,\ 0,\ 0,\ 0,\ 1,\ 0,\ 0,\ 1,\ 1,\ 0,\ 1,\ 0,\ 1)\,, (0,\ 0,\ 1,\ 0,\ 1,\ 0,\ 0,\ 0,\ 0,\ 0,\ 1,\ 1,\ 0,\ 0,\ 0,\ 1)\,,$$
$$(1,\ 0,\ 1,\ 1,\ 1,\ 0,\ 1,\ 1,\ 0,\ 0,\ 1,\ 1,\ 0,\ 0,\ 0,\ 1)\,, (1,\ 1,\ 0,\ 0,\ 0,\ 1,\ 0,\ 1,\ 1,\ 1,\ 0,\ 1,\ 1,\ 0,\ 1,\ 0)]$$

Decodificación

```
yP=[y[i]*P.inverse() for i in range(len(c))]
yd = [decodePatterson(yP[i]) for i in range(len(yP))]
print ("yd")
show(yd)
decodificado=[C.decode_to_message(yd[i])*S.inverse() for i in range(len(yd))]
print ("decodificado")
show(decodificado)
jkk=[list(decodificado[i]) for i  in range(len(decodificado))]
jkk
qqq=[];
for i in range(len(jkk)):
    qqq=qqq+jkk[i]
mensajeDesencriptado=bin_to_ascii(qqq)
print ("mensaje real")
show(mensajeDesencriptado)
```

$$yd = [(1,\ 0,\ 1,\ 1,\ 0,\ 0,\ 0,\ 1,\ 1,\ 0,\ 1,\ 0,\ 1,\ 1,\ 0,\ 1)\,, (1,\ 1,\ 0,\ 0,\ 1,\ 1,\ 1,\ 0,\ 0,\ 1,\ 0,\ 1,\ 0,\ 0,\ 1,\ 1)\,,$$
$$(1,\ 1,\ 0,\ 0,\ 1,\ 1,\ 1,\ 0,\ 0,\ 1,\ 0,\ 1,\ 0,\ 0,\ 1,\ 1)\,, (0,\ 1,\ 0,\ 0,\ 1,\ 1,\ 1,\ 0,\ 0,\ 1,\ 0,\ 1,\ 0,\ 0,\ 1,\ 0)\,,$$
$$(0,\ 1,\ 0,\ 0,\ 1,\ 1,\ 1,\ 0,\ 0,\ 1,\ 0,\ 1,\ 0,\ 0,\ 1,\ 1)\,, (0,\ 0,\ 1,\ 1,\ 0,\ 0,\ 0,\ 1,\ 1,\ 0,\ 1,\ 0,\ 1,\ 1,\ 0,\ 0)]$$

$$decodificado = [(0,\ 1,\ 0,\ 1)\,, (0,\ 1,\ 0,\ 0)\,, (0,\ 1,\ 0,\ 0)\,, (0,\ 1,\ 1,\ 0)\,, (0,\ 1,\ 0,\ 0)\,, (0,\ 1,\ 1,\ 1)]$$

$$mensajeDesencriptado = \texttt{TFG}$$

7

CONCLUSIONES

El objetivo principal de este Trabajo Fin de Grado ha sido el estudio del Criptosistema de McEliece, pero a la vez ha brindado la oportunidad de profundizar en conocimientos, tanto de matemática pura, como pueden ser conceptos de estructuras matemáticas, como de códigos lineales e incluso programación informática.

Su realización ha permitido comprobar la seguridad del Criptosistema de McEliece, probando longitudes del código de Goppa para los que es vulnerable, y también longitudes como $N = 2048$ y $k = 1606$ para las que el criptosistema es seguro incluso a nivel post-cuántico.

El uso de Códigos de Goppa y matrices aleatorias permite crear un Criptosistema de McEliece con estructura aleatoria. Además, con SageMath se ha podido comprobar que con los mismos parámetros y el mismo código de Goppa nunca se obtienen dos salidas del Criptosistema de McEliece iguales.

Para concluir, destacar la importancia que tiene el urgente desarrollo de criptografía post-cuántica pues ya es más que evidente que es necesario avanzar en este sentido antes de que llegue el ordenador cuántico dado que la criptografía actual quedará obsoleta y tendrá que ser sustituida.

Bibliografía

[1] Felipe, Tesis, 2021 https://github.com/dfrafelipe30/Tesis/blob/main/algoritmoMcEliece.ipynb/.

[2] Departamento de Álgebra, Facultad de Matemáticas, Universidad de Sevilla, Teoría de Codigos y Criptografia https://personal.us.es/albertocd/Notas_teoria_TCyC.pdf/

[3] Oswaldo José Pérez Luis, O.P.¿Qué familia de códigos es adecuada para la criptografía basada en códigos?(Trabajo Fin de Grado) Universidad de La Laguan https://riull.ull.es/xmlui/bitstream/handle/915/15749/%C2%BFQue%20familia%20de%20codigos%20es%20adecuada%20para%20la%20criptografia%20basada%20en%20codigos.pdf?sequence=1&isAllowed=y/

[4] Daniel Felipe Rambaut Lemus, D.R. Introducción a la Criptografía post-cuántica basada en teoría de códigos (Trabajo presentado como requisito para optar por el título de Profesional en Matemáticas aplicadas y ciencias de la computación). Escuelas de Ingeniería Ciencia y Tecnología. Matemáticas aplicadas y ciencias de la computación. Universidad del Rosario. Bogota-Colombia https://repository.urosario.edu.co/bitstream/handle/10336/31688/Thesis_Template.pdf?sequence=1/

[5] S. Ling and C. Xing. Coding theory: a first course. Cambridge University Press,2004. http://site.iugaza.edu.ps/mashker/files/coding-theory-a-first-course.pdf

[6] David Moreno Centeno, D.M. Criptografía Post-cuántica: Implementación de McEliece y una nueva versión (Trabajo Fin de Grado) Grado en Ingeniería Informática de Servicios y Aplicaciones. Universidad de Valladolid. https://uvadoc.uva.es/bitstream/handle/10324/38361/TFG-B.1368.pdf?sequence=1&isAllowed=y

[7] Traducción por Héctor Yanajara Parra.Manual de SAGE para principiantes. INSTITUTO TECNOLOGICO DE SONORA https://www.sagemath.org/es/Manual_SAGE_principiantes.pdf

[8] Juan del Carmen Grados Vásquez. Ejemplo Criptosistema McEliece en SAGE. Friday, April 19, 2013. http://juaninf.blogspot.com/2013/04/function-make-div-with-id-mycell-sage.html

[9] Sage. Documentation. (02 de Septiembre) 2022 https://doc.sagemath.org/html/en/index.html

Study and Implementation of the McEliece Cryptosystem with SageMath

Manuel Ramos Montesó
Facultad de Ciencias • Sección de Matemáticas
Universidad de La Laguna
alu0101530468@ull.edu.es

Abstract

Modern cryptography will become vulnerable with the development of quantum computering, given that algorithms capable of breaking cryptosystems on polynomial time are already known. Therefore, there are many scientists racing against time to find new ways of resisting these attacks and thus, protect our privacy.

This Final Degree Project covers some background and implementation of McElieceâs Crypto-System, which may well be resistant to quantum computering. This cryptosystem is based on the linear Gappa Codes.

Three introductory chapters are made for covering the basis that will allow us to develop our cryptosystem. On forth and fifth chapter we will see the implementation on Sage Math. Further study on these chapters will reveal great complexity on breaking the cryptosystem, as the use of structure on its creation makes McElieceâs code similar to a random linear code.

1. Error correcting codes

Let be a non-empty set, $\mathcal{A}$, and let be a n-element word of $\mathcal{A}$. A code is a non-empty set of words, i.e., it is a subset $\mathcal{C} \subseteq \mathcal{A}^n$.

Given a code $\mathcal{C} \subseteq \mathcal{A}^n$, the length of the code is the integer n, and the size of the code is the integer $|\mathcal{C}|$

We define the following code parameters:

- Minimum distance from the code to the number

$$d(\mathcal{C}) = min\{d(x, v) \mid x, v \in \mathcal{C}\}.$$

- Relative minimum distance from the code to the number

$$\delta(\mathcal{C}) = \frac{d(\mathcal{C})}{n}$$

- Information transmission rate from code to number

$$R(\mathcal{C}) = \frac{log_q(|\mathcal{C}|)}{n}$$

- Redundancy from code to number

$$n - log_q(|\mathcal{C}|)$$

When $\mathcal{A}$ is a finite field, $\mathcal{A}^n$ has a vector space structure, and our subset $\mathcal{C} \subseteq \mathcal{A}^n$ we will say that it is a Linear Code.

We will say that a linear code $\mathcal{C} \subseteq \mathbb{F}_q^n$ is a code of type (n, q^k, d)

2. Goppa Codes

The Goppa code is an error-correcting code.

Construction:

Let $L = (a_1, \ldots, a_n) \in \mathbb{F}_{q^m}^n$ be an n-tuple of n distinct elements of $\mathbb{F}_{q^m}$ y $g(x) \in \mathbb{F}_{q^m}[x]$ a polynomial of degree t such that $g(a_j) \neq 0$ $\forall j = 1, \ldots, n$ We define the Goppa code with respect to L and g as

$$\Gamma(L, g) := \left\{ c \in \mathbb{F}_q^n \;\middle|\; \sum_{i=1}^{n} \frac{c_i}{x - a_i} \equiv 0 \mod g(x) \right\},$$

that this is a linear code defined on $\mathbb{F}_q$ of length n.

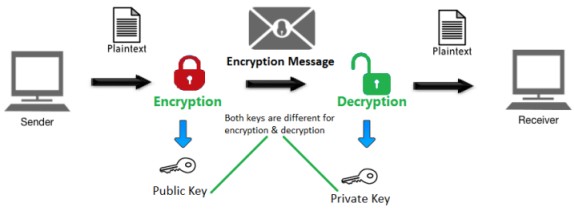
The encoding of the Goppa codes involves dividing the message into blocks of size k and then multiplying each block by the matrix generator G. The resulting vector forms the set of code words, i.e.,

$$(m_1, \ldots, m_k) \times G = (c_1, \ldots, c_n).$$

For decoding we use the Patterson Algorithm. With it, we correct the errors that occur in the transmission.

3. Study of the McEliece cryptosystem

The McEliece cryptosystem is a public key cryptosystem.



It is based on coding theory and its security lies mainly in the fact that the generator matrix of code appears to be random.
To construct such a matrix we first need to build:

- Binary goppa code, and its generator matrix, G
- non-singular square matrix of size k, S
- Permutation matrix of size n, P

Then $G' = S * G * P$

4. Implementation of the McEliece cryptosystem

We have have used the SageMath program to implement the cryptosystem.

TRABAJO FIN DE GRADO, Convocatoria de Septiembre, 2022

Sección de Matemáticas
Universidad de La Laguna