UNIVERSIDAD DE GRANADA

TRABAJO FIN DE GRADO

DOBLE GRADO EN INGENIERÍA INFORMÁTICA Y MATEMÁTICAS

CRIPTOANÁLISIS DEL CRIPTOSISTEMA DE MCELIECE CLÁSICO MEDIANTE ALGORITMOS GENÉTICOS

Autor

PAULA VILLANUEVA NÚÑEZ

Director

GABRIEL NAVARRO GARULO

Facultad de Ciencias

ETSIIT Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

FACULTAD DE CIENCIAS

E.T.S. DE INGENIERÍAS INFORMÁTICA Y DE TELECOMUNICACIÓN

Granada, a 9 de abril de 2022

ÍNDICE GENERAL

RESUMEN

Añadir resumen y palabras clave

PALABRAS CLAVE:

SUMMARY

Añadir resumen y palabras clave en inglés

KEYWORDS:

INTRODUCCIÓN Y OBJETIVOS

Añadir introducción y objetivos

1

PRELIMINARES

En este capítulo se desarrollarán las herramientas necesarias para poder afrontar el criptosistema de McEliece que precisa este trabajo. Empezaremos abordando concepto de anillo y algunas de sus propiedades más importantes. A partir de este noción, podremos definir los cuerpos finitos y los conceptos relacionados con ellos, tales como los anillos de polinomios sobre cuerpos finitos y su utilidad para construir cuerpos finitos. También será de especial relevancia tratar los conceptos relacionados con la teoría de la complejidad para clasificar los problemas de acuerdo a su dificultad. Finalmente, estudiaremos las metaheurísticas basadas en la población para introducir los algoritmos genéticos.

1.1 ANILLOS

En esta sección introduciremos el concepto de anillo para poder definir el concepto de cuerpo, así como las principales propiedades de esta estructura algebraica.

Definición 1. Un *anillo* $(A, +, \cdot)$ es un conjunto A junto con dos operaciones binarias $A \times A \to A$ denotadas por la suma (denotada por $+$) y producto (denotado por $\cdot$) que verifican los siguientes axiomas:

- Propiedad asociativa de la suma:

$$a + (b + c) = (a + b) + c \qquad \forall a, b, c \in A$$

- Existencia del elemento neutro para la suma:

$$0 + a = a = a + 0 \qquad \forall a \in A$$

- Existencia del elemento inverso para la suma:

$$\forall a \in A \ \ \exists - a \in A \quad a + (-a) = 0 = (-a) + a$$

- Propiedad conmutativa de la suma:

$$a + b = b + a \qquad \forall a, b \in A$$

- Propiedad asociativa del producto:

$$a \cdot (b \cdot c) = (a \cdot b) \cdot c \qquad \forall a, b, c \in A$$

- Propiedad distributiva del producto:

$$a \cdot (b + c) = a \cdot b + a \cdot c, \quad (b + c) \cdot a = b \cdot a + c \cdot a \qquad \forall a, b, c \in A$$

Un anillo de llama *conmutativo o abeliano* si se verifica la propiedad conmutativa del producto

$$ab = ba \qquad \forall a, b \in A$$

Es fácil comprobar que en todo anillo se verifica $0 \cdot x = x \cdot 0 = 0$ (obsérvese que $0\cdot = (0 + 0) \cdot x = 0 \cdot x + 0 \cdot x$ y simplifíquese por el simétrico de $0 \cdot x$).

Además del anillo conmutativo, existen otros casos particulares de anillos. Diremos que un anillo es *unitario* si es un anillo cuyo producto tiene elemento neutro, esto es, $\exists 1 \in A \ : \ x \cdot 1 = 1 \cdot x = x$, para todo $x \in A$.

Diremos que un elemento a del anillo A es *invertible* si existe un elemento a' en el anillo A tal que $a \cdot a' = a' \cdot a = 1$. Este elemento a' es único, lo llamaremos *elemento inverso* y lo denotaremos por a^{-1}.

Cuando se da la igualdad $1 = 0$, diremos que el anillo es *trivial* y tendrá un solo elemento.

Definición 2. Sea A un anillo, $1 \in A$ el elemento neutro del producto y $n \geq 1$ un número natural, definimos la cardinalidad de A como:

$$Car(A) = \begin{cases} 0 & \text{si } n \cdot 1 \neq 0 \text{ para cualquier } n \geq 1 \\ n & \text{si } n \text{ es el menor número natural no nulo para el que } n \cdot 1 = 0 \end{cases}$$

1.2 CUERPOS FINITOS

Para presentar el concepto de cuerpo finito, necesitaremos definir previamente el concepto de cuerpo junto con algunas de sus propiedades más relevantes.

Definición 3. Un *cuerpo* $(K, +, \cdot)$ es un anillo conmutativo no trivial en el que todo elemento no nulo tiene un inverso multiplicativo. Se dice que un cuerpo es *finito* si tiene un número finito de elementos.

Sea $(K, +, \cdot)$ un cuerpo y $E \subset K$, diremos que E es un *subcuerpo* de K o K es una *extensión* de E si se cumple que $(E, +, \cdot)$ es un cuerpo cuando las operaciones $+$ y $\cdot$ se restringen a E.

Diremos que la *característica* de un cuerpo es el número de elementos que tiene.

Todos los cuerpos finitos tienen un número de elementos $q = p^n$, para algún número primo p y algún entero positivo n. Denotaremos por $\mathbb{F}_q$ a los cuerpos finitos con característica q, aunque otra común notación es $GF(q)$.

Observemos que si p un número primo y q es un número entero tal que $q = p^n$, entonces $\mathbb{F}_q$ es un espacio vectorial sobre $\mathbb{F}_p$ de dimensión n. Además, hay q vectores en el espacio vectorial de dimensión n sobre $\mathbb{F}_p$.

Notemos que todos los cuerpos finitos de orden q son isomorfos, aunque cada cuerpo puede tener diferentes representaciones.

Proposición 1. *Sea $\mathbb{F}_q$ un cuerpo finito con $q = p^n$ elementos, entonces*

$$p \cdot \alpha = 0, \qquad \forall \alpha \in \mathbb{F}_q.$$

Proposición 2. *Sea $\mathbb{F}_q$ un cuerpo finito con característica p, se cumple que*

$$(\alpha + \beta)^p = \alpha^p + \beta^p, \qquad \forall \alpha, \beta \in \mathbb{F}_q.$$

1.2.1 *Anillo de polinomios sobre cuerpos finitos*

En esta sección vamos a introducir el concepto de polinomio junto con sus operaciones.

Definición 4. Sea A un anillo conmutativo. El *conjunto de polinomios* en la variable x con coeficientes en A está compuesto por el siguiente conjunto

$$\{a_n x^n + a_{n-1}x^{n-1} + \cdots + a_1 x + a_0 \ : \ a_0, ..., a_n \in A\}.$$

Este conjunto se representa por $A[X]$.

En el conjunto de polinomios definimos una suma y un producto.

Sean $f = a_n x^n + a_{n-1}x^{n-1} + \cdots + a_1 x + a_0$ y $g = b_m x^m + b_{m-1}x^{m-1} + \cdots + b_1 x + b_0$ dos polinomios. Supongamos que $m \leq n$, tomando $b_i = 0$ para todo $n \geq i > m$, definimos las operaciones de suma y producto de polinomios:

$$f + g = (a_n + b_n)x^n + (a_{n-1} + b_{n-1})x^{n-1} + \cdots + (a_1 + b_1)x + (a_0 + b_0).$$
$$f \cdot g = a_n b_m x^{n+m} + (a_n b_{m+1} + a_{n-1} b_m)x^{n+m-1} + \cdots + (a_1 b_0 + a_0 b_1)x + a_0 b_0.$$

De esta forma, diremos que el conjunto $A[X]$ con las operaciones anteriores es un *anillo de polinomios en X con coeficientes en A*.

Definición 5. Para un polinomio $f = a_n + a_{n-1}x^{n-1} + \cdots + a_1 x + a_0 \neq 0$ el mayor índice n tal que $a_n \neq 0$ se llama *grado de f* y se representa por $\mathrm{gr}(f)$. Si $f = 0$ definimos $\mathrm{gr}(f) = -\infty$.

Llamaremos *término (de grado i)* a cada uno de los sumandos $a_i X^i$ del polinomio f. El *término líder* es el término no nulo de mayor grado. El coeficiente $a_n \neq 0$ del término líder se llama *coeficiente líder* y el término de grado cero a_0 se llama *término constante*. Si el coeficiente líder es 1, diremos que el polinomio es *mónico*.

A continuación tenemos algunas propiedades de los polinomios.

Proposición 3. *Sea A un anillo conmutativo y sean $f, g \in A[X]$ dos polinomios, tenemos que*

$$\mathrm{gr}(f + g) \leq \mathrm{máx}\,(\mathrm{gr}(f),\ \mathrm{gr}(g)),$$

$$\text{gr}(f \cdot g) \leq \text{gr}(f) + \text{gr}(g)$$

Si $\text{gr}(f) \neq \text{gr}(g)$*, se verifica*

$$\text{gr}(f + g) = \text{máx}\,(\text{gr}(f),\ \text{gr}(g))$$

Podemos trasladar estos resultados al caso de los cuerpos finitos. Sea $\mathbb{F}_q$ un cuerpo finito, un polinomio $f(x)$ estará definido sobre dicho cuerpo si es de la forma $f(x) = \sum_{i=0}^{n} a_i \cdot x^i$, donde $a_i \in \mathbb{F}_q$ para todo $i = 0, ..., n$. Análogamente, diremos que $f(x) \in \mathbb{F}_q$.

Sean $f(x)$ y $g(x)$ polinomios en $\mathbb{F}_q[x]$, diremos que $f(x)$ divide a $g(x)$ si existe un polinomio $h(x) \in \mathbb{F}_q[x]$ tal que $g(x) = f(x)h(x)$ y lo denotaremos por $f(x)|g(x)$. El polinomio $f(x)$ se llama *divisor* o *factor* de $g(x)$.

El *mayor común divisor* de $f(x)$ y $g(x)$ es el polinomio mónico en $\mathbb{F}_q[x]$ con mayor grado que divide a $f(x)$ y a $g(x)$. Este polinomio es único y se denota por $\text{mcd}(f(x),\ g(x))$. Diremos que los polinomios $f(x)$ y $g(x)$ son *primos relativos* si $\text{mcd}(f(x),\ g(x)) = 1$.

El siguiente resultado es de gran utilidad, pues sirve para calcular los divisores de un polinomio e incluso para calcular el máximo común divisor. Nos dará las bases para definir posteriormente el Algoritmo de Euclides.

Teorema 1. *Sean $f(x)$ y $g(x)$ polinomios en $\mathbb{F}_q[x]$ con $g(x)$ no nulo.*

- *Existen dos polinomios únicos $c(x), r(x) \in \mathbb{F}_q[x]$ tales que*

$$f(x) = g(x)c(x) + r(x), \qquad \textit{donde } \text{gr}(r(x)) < \text{gr}(g(x)) \textit{ o } r(x) = 0.$$

- *Si $f(x) = g(x)c(x) + r(x)$, entonces* $\text{mcd}(f(x),\ g(x)) = \text{mcd}(g(x),\ r(x))$.

Los polinomios $c(x)$ y $r(x)$ se llaman *cociente* y *resto*, respectivamente.

Usando este resultado de forma recursiva, obtendremos el máximo común divisor de los polinomios $f(x)$ y $g(x)$. Este procedimiento se conoce como *Algoritmo de Euclides*. El siguiente resultado describe este algoritmo.

Teorema 2 (Algoritmo de Euclides)**.** *Sean $f(x)$ y $g(x)$ polinomios definidos en $\mathbb{F}_q[x]$ con $g(x)$ no nulo.*

1. *Realizar los siguientes pasos hasta que $r_n(x) = 0$ para algún n:*

$$\begin{aligned}
f(x) &= g(x)c_1(x) + r_1(x), && \textit{donde } \text{gr}(r_1(x)) < \text{gr}(x),\\
g(x) &= r_1(x)c_2(x) + r_2(x), && \textit{donde } \text{gr}(r_2(x)) < \text{gr}(r_1),\\
r_1(x) &= r_2(x)c_3(x) + r_3(x), && \textit{donde } \text{gr}(r_3(x)) < \text{gr}(r_2),\\
&\vdots\\
r_{n-3}(x) &= r_{n-2}(x)c_{n-1}(x) + r_{n-1}(x), && \textit{donde } \text{gr}(r_{n-1}(x)) < \text{gr}(r_{n-2}),\\
r_{n-2}(x) &= r_{n-1}(x)c_n(x) + r_n(x), && \textit{donde } r_n(x) = 0.
\end{aligned}$$

   Entonces $\text{mcd}(f(x),\ g(x)) = cr_{n-1}(x)$*, donde $c \in \mathbb{F}_q$ es una constante para que $cr_{n-1}(x)$ sea mónico.*

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

$$a(x)f(x) + b(x)g(x) = \text{mcd}(f(x),\ g(x)).$$

En cada paso el grado del resto se decrementa al menos en 1, por lo que podemos asegurar que la secuencia de pasos anterior terminará en algún momento.

A continuación se muestran algunos resultados relevantes.

Proposición 4. *Sean* $f(x)$ *y* $g(x)$ *polinomios en* $\mathbb{F}_q[x]$.

- *Si* $k(x)$ *es un divisor de* $f(x)$ *y* $g(x)$, *entonces* $k(x)$ *es un divisor de* $a(x)f(x) + b(x)g(x)$ *para algunos* $a(x), b(x) \in \mathbb{F}_q[x]$.
- *Si* $k(x)$ *es un divisor de* $f(x)$ *y* $g(x)$, *entonces* $k(x)$ *es un divisor de* $\text{mcd}(f(x),\ g(x))$.

Proposición 5. *Sea* $f(x)$ *un polinomio en* $\mathbb{F}_q[x]$ *de grado* n.

- *Si* $\alpha \in \mathbb{F}_q$ *es una raíz de* $f(x)$, *entonces* $x - \alpha$ *es un factor de* $f(x)$.
- *El polinomio* $f(x)$ *tiene como mucho* n *raíces en cualquier cuerpo que contenga a* $\mathbb{F}_q$.

Teorema 3. *Los elementos de* $\mathbb{F}_q$ *son las raíces de* $x^q - x$.

1.2.1.1 *Construcción de cuerpos finitos*

Para realizar la construcción de cuerpos finitos, previamente necesitaremos conocer el siguiente concepto y algunos resultados relacionados.

Definición 6. Sea $f(x) \in \mathbb{F}_q[x]$ un polinomio no constante, decimos que es *irreducible* sobre $\mathbb{F}_q$ si no se factoriza como producto de dos polinomios en $\mathbb{F}_q[x]$ de menor grado.

Teorema 4. *Sea* $f(x)$ *un polinomio no constante. Entonces*

$$f(x) = p_1(x)^{a_1} \cdots p_k(x)^{a_k},$$

donde cada $p_i(x)$ *es irreducible y único salvo orden, y los elementos* a_i *son únicos.*

Como consecuencia de este resultado, tenemos que $\mathbb{F}_q[x]$ es un *dominio de factorización única*.

El siguiente resultado nos muestra cómo construir un cuerpo finito de característica p a partir del cociente de anillos de polinomios por polinomios irreducibles.

Proposición 6. *Sea* p *un número primo y sea el polinomio* $f(x) \in \mathbb{F}_p[x]$ *irreducible en* $\mathbb{F}_p$ *y de grado* m. *Tenemos que el anillo cociente* $\mathbb{F}_q[x]/\ (f(x))$ *es un cuerpo finito con* $q = p^m$ *elementos, es decir, con característica* p.

Escribiremos los elementos del anillo cociente, que son las clases laterales $g(x) + (f(x))$ como vectores en $\mathbb{F}_p^m$ con la siguiente correspondencia:

$$g_{m-1}x^{m-1} + g_{m-2}x^{m-2} + \cdots + g_1x + g_0 + (f(x)) \leftrightarrow (g_{m-1}, g_{m-2}, ..., g_1, g_0). \tag{1}$$

Esta notación facilita la operación de sumar dos elementos. Sin embargo, la multiplicación es algo más complicada. Supongamos que queremos multiplicar $g_1(x) + (f(x))$ por $g_2(x) + (f(x))$. Para ello, usaremos el resultado 1 para obtener

$$g_1(x)g_2(x) = f(x)h(x) + r(x), \tag{2}$$

donde $\text{gr}(r(x)) \leq m - 1$ o $r(x) = 0$. Entonces

$$(g_1(x) + (f(x)))(g_2(x) + (f(x))) = r(x) + (f(x)).$$

Podemos simplificar la notación si reemplazamos x por α donde $f(\alpha) = 0$. Por 2, se cumple que $g_1(\alpha)g_2(\alpha) = r(\alpha)$ y la correspondencia 1 queda como sigue

$$g_{m-1}g_{m-2}\cdots g_1g_0 \leftrightarrow g_{m-1}\alpha^{m-1}g_{m-2}\alpha^{m-2}\cdots g_1\alpha g_0.$$

De esta forma, multiplicamos los polinomios en α de forma usual y aplicamos al resultado que $f(\alpha) = 0$ para reducir las potencias de α mayores que $m - 1$ a polinomios en α de grado menor que m.

1.3 TEORÍA DE LA COMPLEJIDAD

El objetivo de esta sección se basa en clasificar los problemas de acuerdo a su dificultad. Nos centraremos en estudiar la complejidad de los problemas decidibles, es decir, los problemas que pueden tener algoritmos para resolverlos.

1.3.1 *Complejidad de los algoritmos*

A la hora de resolver un problema con un algoritmo, tendrán gran importancia los recursos que consume dicho algoritmo, sobre todo el espacio y el tiempo. La complejidad del tiempo de un algoritmo es el número de pasos que necesita para resolver un problema de tamaño n. La complejidad de un algoritmo siempre se define en el peor de los casos, esto es, se obtiene una cota para el recuento de los pasos en el peor de los casos.

La siguiente definición nos proporciona la notación más importante en el análisis de los algoritmos.

Definición 7 (Notación $\mathcal{O}$-grande)**.** Sean $f, g : \mathbb{N}^r \longrightarrow \mathbb{R}$. Decimos que $f(n) = O(g(n))$ si existen dos constantes $c \in \mathbb{R}^+$ y $n_0 \in \mathbb{N}^r$ tales que $\forall n \geq n_0$, $f(n) \leq c \cdot g(n)$.

Con esta notación podemos analizar el tiempo esperado que va a tardar un algoritmo en realizar una tarea.

Proposición 7. *Para cualquier $\lambda \in \mathbb{R}^+$ y $g : \mathbb{N}^r \longrightarrow \mathbb{R}$, tenemos que*

$$\mathcal{O}(g(n)) = \mathcal{O}(\lambda g(n)), \qquad \forall n \in \mathbb{N}^r.$$

A continuación presentaremos algunos conceptos importantes para especificar la complejidad de los algoritmos que nos serán de utilidad más adelante.

Definición 8 (Algoritmo en tiempo polinomial)**.** Un algoritmo es un *algoritmo en tiempo polinomial* si su complejidad es $\mathcal{O}(p(n))$, donde $p(n)$ es una función polinómica de n.

Definición 9 (Algoritmo en tiempo exponencial)**.** Un algoritmo es un *algoritmo en tiempo exponencial* si su complejidad es $\mathcal{O}(c^n)$, donde c es una constante real estrictamente mayor que 1.

1.3.2 *Complejidad de los problemas*

La complejidad de un problema es equivalente a la complejidad del mejor algoritmo que resuelve ese problema. Diremos que un problema es *fácil* si existe un algoritmo en tiempo polinomial que lo resuelve, mientras que un problema será *difícil* si no existe ningún algoritmo en tiempo polinomial que lo resuelva.

La teoría de la complejidad de los problemas se ocupa de los *problemas de decisión*. Un problema de decisión siempre tiene una respuesta de sí o no.

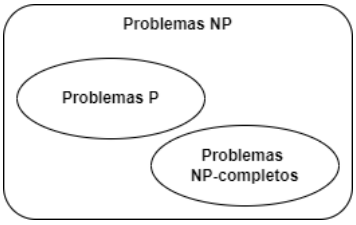


Figura 1: Clases de complejidad de los problemas de decisión.

Con la teoría computacional podemos clasificar los problemas en sus clases de complejidad. Existen dos clases importantes de problemas: P y NP (Figura 1).

La clase de complejidad P representa a la familia de problemas de decisión que se pueden resolver mediante un algoritmo en tiempo polinomial. Los problemas que pertenecen a esta clase son *fáciles* de resolver.

La clase de compejidad NP representa a la familia de problemas de decisión que se pueden resolver por un algoritmo no determinista en tiempo polinomial. Un algoritmo no determinista tiene algunos puntos en los que hay múltiples combinaciones diferentes sin ninguna especificación de cuál se tomará. Estos problemas se dicen que son más *difíciles* de resolver.

Para definir la clase de los problemas NP-completos, necesitaremos previamente definir el concepto de reducción.

Definición 10 (Reducción polinómica)**.** Un problema de decisión A se *reduce polinómicamente* a otro problema de decisión B si, para todas las instancias de entrada I_A de A, siempre se

puede construir una instancia de entrada I_B para B en una función de tiempo polinomial, de tal forma que I_A es una instancia positiva de A si y solo si I_B es una instancia positiva de B.

Un problema de decisión es *NP-completo* si es NP y cualquier otro problema de NP se reduce a él. Los problemas que pertenecen a esta clase son los *más difíciles* dentro de la clase NP.

La figura 1 muestra la relación entre los problemas P, NP y NP-completos.

Los problemas *NP-duros* son problemas de optimización cuyo problema de decisión asociado es NP-completo. Las metaheurísticas constituyen una importante alternativa para resolver esta clase de problemas.

1.4 METAHEURÍSTICAS BASADAS EN LA POBLACIÓN

Las *metaheurísticas* representan una familia de técnicas de optimización aproximada. Proporcionan soluciones aceptables en un tiempo razonable para resolver problemas difíciles y complejos, esto es, problemas que no pertenecen a la clase de complejidad P o que no se conoce que pertenezcan a dicha clase. Sin embargo, no garantizan la optimalidad de las soluciones obtenidas.

Las *metaheurísticas basadas en la población* se pueden ver como una mejora iterativa en una población de soluciones. Estas metaheurísticas consisten en inicializar la población, posteriormente generan una nueva población de soluciones y finalmente esta nueva población reemplaza a la actual usando algunos procedimientos de selección. Una vez se alcanza la condición dada, el proceso de búsqueda finaliza y obtenemos una solución.

El esquema de este procedimiento se muestra a continuación.

Output: Mejor solución encontrada
1 $P \longleftarrow P_0$ // Generación de la población inicial
2 $t \longleftarrow 0$
3 **while** *No se cumpla la condición de parada* **do**
4 Generar(P'_t) // Generación de la nueva población
5 $P_{t+1} \longleftarrow$ SeleccionarPoblación($P_t \cup P'_t$) // Selección de la nueva población
6 $t \longleftarrow t + 1$
7 **end**

Vamos a estudiar en detalle este procedimiento concretando los conceptos en los que se basa.

1. **Generación de la población inicial**. Las metaheurísticas basadas en la población comienzan con una población inicial de soluciones. Este paso juega un papel fundamental en la efectividad del algoritmo y su eficiencia. El principal criterio para generar la población es que haya diversidad.
2. **Generación de una nueva población**. En este paso, se genera una nueva población de soluciones. Se puede realizar de varias formas, sin embargo la que estudiaremos será la que se basa en la evolución. En esta categoría, las soluciones que componen la población

se seleccionan y se reproducen usando operadores de variación (por ejemplo, mutación, recombinación) y actúan directamente sobre sus representaciones. Una nueva solución surge a partir de los diferentes atributos de las soluciones de la población actual.

3. **Selección de la nueva población**. Este último paso consiste en seleccionar las nuevas soluciones y reemplazar a la población actual. Este reemplazamiento se puede realizar de varias maneras, por ejemplo una estrategia se basa en seleccionar la nueva población generada como la población actual, otra estrategia se fundamenta en el escoger las dos mejores soluciones de la nueva población y reemplazarlas por las dos peores de la población actual, etc.

El criterio de parada suele ser un número máximo de iteraciones (generaciones), un número máximo de evaluaciones de la función objetivo, etc.

1.4.1 *Algoritmos evolutivos*

Los *algoritmos evolutivos* son algoritmos basados en poblaciones. Estos se fundamentan en el concepto de *competición*. Se representan como una clase de algoritmos de optimización iterativos que simulan la evolución de las especies, en este caso, la evolución de la población de soluciones.

Inicialmente, tenemos una población de soluciones que suele ser generada aleatoriamente. También tenemos una función objetivo, que es la que queremos optimizar, que evalúa cada individuo perteneciente a la población indicando la idoneidad de pertenencia al problema (*fitness*). En cada paso, los individuos se seleccionan para formar los padres. En esta selección los individuos con mayor fitness tendrán altas probabilidades de ser escogidos. Luego, los individuos seleccionados se reproducen usando operadores de variación (por ejemplo, cruce, mutación) para generar nuevos descendientes, de forma que nos permite explorar el espacio de búsqueda y obtendremos soluciones diversas. Finalmente, el esquema de reemplazamiento se basa en determinar los individuos que sobrevivirán entre los descendientes y los padres. Esta iteración representa una generación. Este proceso es iterativo hasta que se cumpla el criterio de parada.

Cada solución que forma parte de la población se suele llamar *cromosoma*. A su vez, cada cromosoma está formado por *genes*.

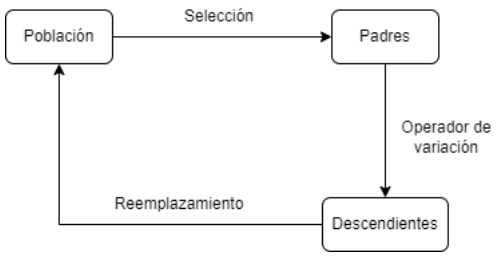


Figura 2: Generación en algoritmos evolutivos.

El algoritmo 2 ilustra este procedimiento.

Output: Mejor solución encontrada
1 Generar(P_0) // Generación de la población inicial
2 $t \longleftarrow 0$
3 **while** *No se cumpla la condición de parada* **do**
4 Evaluar(P_t) $P'_t \longleftarrow$ Seleccionar(P_t)
5 $P'_t \longleftarrow$ Reproducción(P'_t)
6 Evaluar(P'_t)
7 $P_{t+1} \longleftarrow$ Reemplazar(P_t, P'_t)
8 $t \longleftarrow t + 1$
9 **end**

1.4.1.1 *Algoritmos genéticos*

Los *algoritmos genéticos* son algoritmos de optimización, búsqueda y aprendizaje inspirados en los procesos de evolución natural y evolución genética.

Estos algoritmos suelen aplicar como operador de variación un operador de cruce a las dos soluciones (cromosomas) que juegan un papel importante, además de un operador de mutación que modifica aleatoriamente el contenido del individuo (gen) para fomentar la diversidad. Los algoritmos genéticos usan una selección probabilística. El reemplazamiento es generacional, esto es, los padres serán reemplazados por sus descendientes. El operador de cruce se basa en un cruce uniforme o por segmento fijo, mientras que la mutación, con cierta probabilidad, determinará si algún gen puede mutar.

El operador de cruce es un operador binario y, a veces, n-ario. Este operador se basa en adquirir las características de los dos padres y generar un descendiente. La principal característica de este operador es la *heredabilidad*, pues crea un cromosoma a partir de los genes heredados de los padres. Esto es, el descendiente tendrá una recombinación de las características de sus padres. Los padres cruzarán con una cierta probabilidad p_c. El operador de cruce debe producir soluciones válidas.

El operador de mutación es un operador unario que actúa sobre un solo individuo. Las mutaciones representan pequeños cambios de los individuos seleccionados. La probabilidad p_m define la probabilidad de mutar cada elemento (gen) de la representación. Este operador debe garantizar que cualquier solución estará en el espacio de búsqueda y que sea válida. Con la mutación conseguimos introducir diversidad en los individuos de una población.

El algoritmo 3 ilustra este procedimiento.

Output: Mejor solución encontrada

```
1 Generar(P_0) // Generación de la población inicial
2 t ⟵ 0
3 while No se cumpla la condición de parada do
4 |   Evaluar(P_t) P'_t ⟵ Seleccionar(P_t)
5 |   P'_t ⟵ OperadorCruce(P'_t)
6 |   P'_t ⟵ OperadorMutación(P'_t)
7 |   P_{t+1} ⟵ Reemplazar(P_t, P'_t)
8 |   t ⟵ t + 1
9 end
```

Este modelo es elitista y produce una convergencia rápida cuando se reemplazan los peores cromosomas de la población.

2

INTRODUCCIÓN A LA TEORÍA DE CÓDIGOS LINEALES

En este capítulo introduciremos los conceptos y resultados fundamentales sobre la teoría de códigos lineales. Empezaremos con la definiendo básica de código para estudiar posteriormente el concepto de código lineal y algunas de sus propiedades más relevante, junto con otros conceptos relacionados. Finalmente, introduciremos el algoritmo de Brouwer-Zimmermann, que permite calcular un concepto muy importante en la teoría de códigos.

El desarrollo de este capítulo se ha basado, principalmente, en [1, Capítulo 1].

2.1 INTRODUCCIÓN

Supongamos que queremos enviar un mensaje, por lo que habrá un emisor y un receptor que se comunican, en general, en una dirección. Este mensaje es una secuencia finita de elementos de un alfabeto dado y es enviado por un *canal de comunicación*, a través del cual es posible que la información se altere por las interferencias y el ruido, lo que se conoce como *ruido del canal*. Es por esto que hay que hacer una *traducción* entre el mensaje original (o *palabra fuente*) x y el tipo de mensaje c que el canal está capacitado para enviar (*palabras código*). Esta manipulación consiste en proteger el mensaje original, ya sea por ejemplo añadiendo redundancia o repitiéndolo, para que posteriormente se pueda corregir el ruido hasta cierto punto. Este proceso se llama *codificación*. Una vez codificado el mensaje, lo enviamos a través del canal, y nuestro intermediario (el receptor) recibe un mensaje codificado (*palabra recibida*) posiblemente erróneo. Una vez recibido, empieza el proceso llamado *corrección de errores*, que consiste en recuperar el mensaje original corrigiendo los errores que se hubieran producido. El mensaje recibido c' es traducido nuevamente a términos originales x', es decir, es *decodificado*. La siguiente figura representa un esquema de este proceso.

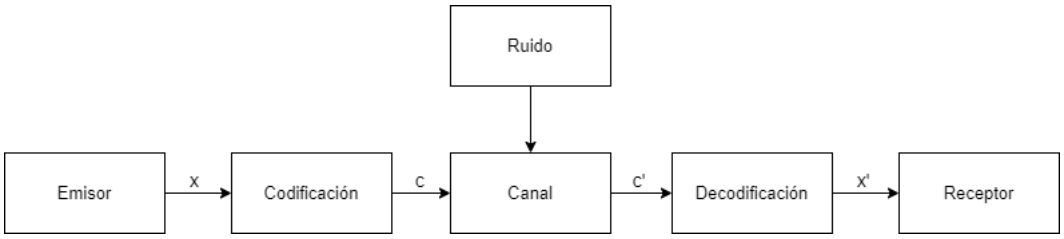


Figura 3: Esquema del modelo de comunicación, [2].

Las flechas indican que la comunicación es en un solo sentido.

En general, $x' \neq x$ y es deseable que este error sea detectado (lo cual permite pedir una retransmisión del mensaje) y en lo posible corregido.

La *Teoría de Códigos Autocorrectores* se ocupa del segundo y cuarto pasos del esquema anterior, es decir, de la codificación y decodificación de mensajes, junto con el problema de detectar y corregir errores. A veces no es posible pedir retransmisión de mensajes y es por eso que los códigos autocorrectores son tan útiles y necesarios.

La calidad de un código con mensajes de longitud k y palabras código de longitud n vendrá dada por las siguientes características.

- El cociente $\frac{k}{n}$, el *ratio de información* del código, que mide el esfuerzo necesario para transmitir un mensaje codificado.
- La *distancia mínima relativa d* que es aproximadamente el doble de la proporción de errores que se pueden corregir en cada mensaje codificado.
- La *complejidad* de los procedimientos de codificar y decodificar.

De esta forma, uno de los objetivos centrales de la teoría de códigos autocorrectores es construir códigos que sean de calidad. Esto es, códigos que permitan codificar muchos mensajes, que se puedan trasmitir rápida y eficientemente, que detecten y corrijan simultáneamente la mayor cantidad de errores posibles y que haya algoritmos de decodificación eficientes y efectivos. Por lo que habrá que encontrar un balance entre estas distintas metas, pues suelen ser contradictorias entre sí.

2.2 CÓDIGOS LINEALES

Sea F_q el cuerpo finito con q elementos, denotamos por $\mathbb{F}_q^n$ el espacio vectorial de las n-tuplas sobre el cuerpo finito $\mathbb{F}_q$. Generalmente los vectores $(a_1, ..., a_n)$ de $\mathbb{F}_q^n$ se denotarán por $a_1 \cdots a_n$.

Definición 11. Un (n, M) *código* $\mathcal{C}$ sobre $\mathbb{F}_q$ es un subconjunto de $\mathbb{F}_q^n$ de tamaño M. A los elementos de $\mathcal{C}$ los llamaremos *palabras código*.

Ejemplo 1.

- Un código sobre $\mathbb{F}_2$ se llama *código binario* y un ejemplo es $\mathcal{C} = \{00, 01, 10, 11\}$.
- Un código sobre $\mathbb{F}_3$ se llama *código ternario* y un ejemplo es $\mathcal{C} = \{21, 02, 10, 20\}$.

Si $\mathcal{C}$ es un subespacio k-dimensional de $\mathbb{F}_q^n$, entonces decimos que $\mathcal{C}$ es un $[n, k]$ *código lineal* sobre $\mathbb{F}_q$. De esta forma, los códigos lineales tendrán q^k palabras código. Al imponer linealidad sobre los códigos, nos permite conseguir algoritmos de codificación y decodificación más eficientes que otros códigos. Estos se pueden presentar con una matriz generadora o con una matriz de paridad.

Definición 12. Una *matriz generadora* para un $[n, k]$ código $\mathcal{C}$ es una matriz $k \times n$ donde sus filas forman una base de $\mathcal{C}$.

Definición 13. Para cada conjunto de k columnas independientes de una matriz generadora G, se dice que el conjunto de coordenadas correspondiente conforman un *conjunto de información* de $\mathcal{C}$. Las $r = n - k$ restantes coordenadas se denominan *conjunto de redundancia* y el número r es la *redundancia* de $\mathcal{C}$.

En general, la matriz generadora no es única pues si realizamos un cambio de base del código podemos obtener otra matriz generadora distinta. Sin embargo, si las k primeras coordenadas conforman un conjunto de información, entonces el código tiene una única matriz generadora de la forma $(I_k|A)$, donde I_k denota a la matriz identidad $k \times k$. Esta matriz se dice que está en *forma estándar*.

Como un código lineal es un subespacio de un espacio vectorial, es el núcleo de alguna transformación lineal.

Definición 14. Una *matriz de paridad* H de dimensión $(n-k) \times n$ de un $[n,k]$ código $\mathcal{C}$ es una matriz que verifica que

$$C = \left\{ \mathbf{x} \in \mathbb{F}_q^n : H\mathbf{x}^T = 0 \right\}.$$

Al igual que con la matriz generadora, la matriz de paridad no es única. Con el siguiente resultado podremos obtener una matriz de paridad cuando $\mathcal{C}$ tiene una matriz generadora en forma estándar.

Teorema 5. *Si $G = (I_k|A)$ es una matriz generadora para el $[n,k]$ código $\mathcal{C}$ en forma estándar, entonces $H = \left(-A^T|I_{n-k}\right)$ es una matriz de paridad de $\mathcal{C}$.*

Demostración. Como $HG^T = -A^T + A^T = 0$, se tiene que $\mathcal{C}$ está contenido en el núcleo de la transformación lineal $x \mapsto Hx^T$. Esta transformación lineal tiene un núcleo de dimensión k, pues H tiene rango $n-k$, que coincide con la dimensión de $\mathcal{C}$. □

Ejemplo 2. Sea la matriz $G = (I_4|A)$, donde

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

es una matriz generadora en forma estándar para un $[7,4]$ código binario que denotaremos por $\mathcal{H}_3$. Por el Teorema 5, una matriz de paridad de $\mathcal{H}_3$ es

$$H = \left(\begin{array}{c|c} -A^T & I_{7-4} \end{array}\right) = \left(\begin{array}{c|c} -A^T & I_3 \end{array}\right) = \left(\begin{array}{cccc|ccc} 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{array}\right)$$

Este código se denomina el $[7,4]$ *código de Hamming*.

2.3 CÓDIGO DUAL

Sabemos que $\mathcal{C}$ es un subespacio de un espacio vectorial, por lo que podemos calcular el subespacio ortogonal a dicho subespacio y así obtener lo que se denomina *espacio dual u ortogonal* de $\mathcal{C}$, denotado por $\mathcal{C}^\perp$. Se define este concepto con la operación del producto escalar como sigue.

Definición 15. El *espacio dual* de $\mathcal{C}$ viene dado por

$$\mathcal{C}^\perp = \left\{ \mathbf{x} \in \mathbb{F}_q^n \ : \ \mathbf{c} \cdot \mathbf{x} = 0 \quad \forall \mathbf{c} \in \mathcal{C} \right\}$$

El siguiente resultado nos muestra cómo obtener las matrices generadora y de paridad de $\mathcal{C}^\perp$ a partir de las de $\mathcal{C}$.

Proposición 8. *Si tenemos una matriz generadora G y una matriz de paridad H de un código $\mathcal{C}$, entonces H y G son matrices generadoras y de paridad, respectivamente, de $\mathcal{C}^\perp$.*

Demostración. Sea G una matriz generadora y H una matriz de paridad de un código $\mathcal{C}$. Sabemos que $G \cdot H^T = 0$. Por otra parte, tenemos que

$$\mathcal{C}^\perp = \left\{ \mathbf{x} \in \mathbb{F}_q^n \ : \ \mathbf{c} \cdot \mathbf{x} = 0 \quad \forall \mathbf{c} \in \mathcal{C} \right\} = \left\{ \mathbf{x} \in \mathbb{F}_q^n \ : \ G \cdot \mathbf{x}^T = 0 \quad \forall \mathbf{c} \in \mathcal{C} \right\}.$$

Luego la matriz H es una matriz generadora de $\mathcal{C}^\perp$.

Además, como $H \cdot G^T = 0$, entonces G es una matriz de paridad de $\mathcal{C}^\perp$. □

De la proposición anterior se deduce lo siguiente.

Proposición 9. *$\mathcal{C}^\perp$ es un $[n, n-k]$ código.*

Demostración. Sabemos que, por ser G una matriz de paridad de $\mathcal{C}^\perp$,

$$\mathcal{C}^\perp = \left\{ x \in \mathbb{F}_q^k : Gx^T = 0 \right\},$$

o sea $\mathcal{C}^\perp$ es el espacio solución de k ecuaciones con n incógnitas. Luego, como G tiene rango k, hay $n-k$ variables libres, por lo tanto $\dim \mathcal{C}^\perp = n-k$. □

Diremos que un código $\mathcal{C}$ es *auto-ortogonal* si $\mathcal{C} \subseteq \mathcal{C}^\perp$ y *auto-dual* cuando $\mathcal{C} = \mathcal{C}^\perp$.

Ejemplo 3. Una matriz generadora para el $[7,4]$ código de Hamming $\mathcal{H}_3$ se presenta en el Ejemplo 2. Sea $\mathcal{H}'_3$ el código de longitud 8 y dimensión 4 obtenido de $\mathcal{H}_3$ añadiendo una coordenada de verificación de paridad general a cada vector de G y por lo tanto a cada palabra código de $\mathcal{H}_3$. Entonces

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

es una matriz generadora para $\mathcal{H}'_3$. Además, veamos que $\mathcal{H}'_3$ es un código auto-dual.

Tenemos que $G' = (I_4|A')$, donde

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

Como $A'(A')^T = I_4$, entonces $\mathcal{H}'_3$ es auto-dual.

2.4 PESOS Y DISTANCIAS

A la hora de corregir errores es importante establecer una medida que nos establezca cuánto de diferentes son las palabras enviadas y recibidas. En este apartado estudiaremos esta idea y cómo puede influir a la teoría de códigos.

Definición 16. La *distancia de Hamming* $\mathrm{d}(\mathbf{x}, \mathbf{y})$ entre dos vectores $\mathbf{x}, \mathbf{y} \in \mathbb{F}_q^n$ se define como el número de coordenadas en las que $\mathbf{x}$ e $\mathbf{y}$ difieren.

Ejemplo 4. Sean $\mathbf{x} = 012$, $\mathbf{y} = 210$ $\mathbf{x}, \mathbf{y} \in \mathbb{F}_3^4$. Entonces la distancia de Hamming entre los dos vectores es $d(\mathbf{x}, \mathbf{y}) = 2$.

Teorema 6. *La función distancia* $\mathrm{d}(\mathbf{x}, \mathbf{y})$ *satisface las siguientes propiedades.*

1. *No negatividad:* $\mathrm{d}(\mathbf{x}, \mathbf{y}) \geq 0 \quad \forall \mathbf{x}, \mathbf{y} \in \mathbb{F}_q^n$.
2. $\mathrm{d}(\mathbf{x}, \mathbf{y}) = 0 \Leftrightarrow \mathbf{x} = \mathbf{y}$.
3. *Simetría:* $\mathrm{d}(\mathbf{x}, \mathbf{y}) = \mathrm{d}(\mathbf{y}, \mathbf{x}) \quad \forall \mathbf{x}, \mathbf{y} \in \mathbb{F}_q^n$.
4. *Desigualdad triangular:* $\mathrm{d}(\mathbf{x}, \mathbf{z}) \leq \mathrm{d}(\mathbf{x}, \mathbf{y}) + \mathrm{d}(\mathbf{y}, \mathbf{z}) \quad \forall \mathbf{x}, \mathbf{y}, \mathbf{z} \in \mathbb{F}_q^n$

Demostración. Las tres primeras afirmaciones se obtienen directamente a partir de la definición. La cuarta propiedad se obtiene a partir de la no negatividad. Esto es, sean $x, y, z \in \mathbb{F}_q^n$ distingamos dos casos. Si $x \neq z$ no puede ocurrir que $x = y = z$, luego $y \neq x$ o $y \neq z$. Es decir, se cumple que $d(x, y) \neq 0$ o $d(y, z) \neq 0$. Entonces por la no negatividad se da la desigualdad. En el caso en el que $x = z$, tendríamos que $d(x, z) = 0$ y también se da la afirmación. □

Diremos que la *distancia mínima* de un código $\mathcal{C}$ es la distancia más pequeña de todas las distancias entre dos palabras distintas del código. Esta medida es fundamental a la hora de determinar la capacidad de corregir errores de $\mathcal{C}$.

Ejemplo 5. Sea $\mathcal{C} = \{010101, 212121, 111000\}$ un código ternario. Entonces

$$d(010101, 212121) = 3, \qquad d(010101, 111000) = 4, \qquad d(212121, 111000) = 5.$$

Por lo que la distancia mínima del código $\mathcal{C}$ es $d(\mathcal{C}) = 3$.

Teorema 7 (Decodificación de máxima verosimilitud)**.** *Es posible corregir hasta*

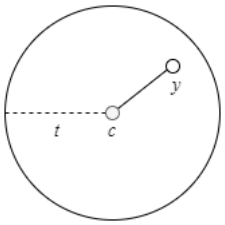
$$t := \left\lfloor \frac{d(\mathcal{C}) - 1}{2} \right\rfloor$$

errores, donde $d(\mathcal{C})$ denota la distancia mínima del código $\mathcal{C}$.

Demostración. Usando la decodificación de máxima verosimilitud, un vector $y \in \mathbb{F}^n$ es decodificado en una palabra código $c \in \mathcal{C}$, que es cercana a y con respecto a la distancia de Hamming. Formalmente, y es decodificado en una palabra código $c \in \mathcal{C}$ tal que $d(c, y) \leq d(c', y)$, $\forall c' \in \mathcal{C}$. Si hay varios $c \in \mathcal{C}$ con esta propiedad, se elige uno arbitrariamente.

Si la palabra código $c \in \mathcal{C}$ fue enviada y no han ocurrido más de t errores durante la transmisión, el vector recibido es

$$y = c + e \in \mathbb{F}^n,$$



donde e denota al vector error.

Esto satisface

$$d(c, y) = d(e, 0) \leq t,$$

y por lo tanto c es el único elemento de $\mathcal{C}$ que se encuentra en una bola de radio t alrededor de y. Un decodificador de máxima verosimilitud produce este elemento c, y así se obtiene el código correcto. □

Definición 17. El *peso Hamming* $\mathrm{wt}(\mathbf{x})$ de un vector $\mathbf{x} \in \mathbb{F}_q^n$ se define como el número de coordenadas no nulas en $\mathbf{x}$.

Ejemplo 6. Sea $\mathbf{x} = 2001021 \in \mathbb{F}_3^7$ un vector, entonces su peso Hamming es $wt(\mathbf{x}) = 4$.

El siguiente resultado nos muestra la relación entre la distancia y el peso.

Teorema 8. *Si $\mathbf{x}, \mathbf{y} \in \mathbb{F}_q^n$, entonces $\mathrm{d}(\mathbf{x}, \mathbf{y}) = \mathrm{wt}(\mathbf{x} - \mathbf{y})$. Si $\mathcal{C}$ es un código lineal, entonces la distancia mínima d coincide con el peso mínimo de las palabras código no nulas de $\mathcal{C}$.*

Demostración. Sean $x, y \in \mathbb{F}_q^n$, por la definición de distancia de Hamming tenemos que $d(x, y) = wt(x - y)$. Se supone ahora que C es un código lineal, luego para todo $x, y \in \mathcal{C}$, $x - y \in \mathcal{C}$, luego para cualquier par de elementos $x, y \in \mathcal{C}$, existe $z \in \mathcal{C}$ tal que $d(x, y) = wt(z) \geq wt(\mathcal{C})$, donde $wt(\mathcal{C})$ es el peso mínimo de $\mathcal{C}$. Por tanto, $d \geq wt(\mathcal{C})$. Por otro lado, para todo $x \in \mathcal{C}$, se tiene que $wt(x) = d(x, 0)$. Como $\mathcal{C}$ es lineal, $0 \in \mathcal{C}$, luego $d(x, 0) \geq d$. Entonces, $wt(\mathcal{C}) \geq d$. Se concluye que $wt(\mathcal{C}) = d$, como se quería. □

Como consecuencia de este teorema, para códigos lineales, la distancia mínima también se denomina *peso mínimo* de un código. Si se conoce el peso mínimo d de un $[n, k]$ código, se dice entonces que es un $[n, k, d]_q$ código.

En el artículo [3] se ha demostrado que el problema de calcular la distancia mínima de un código lineal binario es NP-duro, y el problema de decisión correspondiente es NP-completo.

Problema 1 (Distancia mínima)**.** *Dada una matriz binaria H de dimensión $m \times n$ y un número entero $w > 0$. ¿Existe un vector no nulo $x \in \mathbb{F}_2^n$ de peso menor que w tal que $Hw^T = 0$?*

El problema 1 es un problema NP-completo. Veamos un esquema de la demostración de este resultado. Para ello, haremos uso de una transformación polinomial del problema de Decodificación de Máxima Verosimilitud al problema de Distancia Mínima.

Problema 2 (Decodificación de Máxima Verosimilitud)**.** *Dados una matriz binaria H de dimensión $m \times n$, un vector $s \in \mathbb{F}_2^m$ y un número entero $w > 0$. ¿Existe un vector $x \in \mathbb{F}_2^n$ de peso menor o igual que w tal que $Hx^t = s$?*

El problema 2 es NP-completo y sigue siendo NP-completo bajo ciertas restricciones, luego reformularemos este problema como la versión de campo finito de Suma de Subconjuntos, un problema NP-completo conocido. Además, calcular la distancia mínima para la clase de códigos lineales sobre un cuerpo de característica 2 es NP-duro, y el problema de decisión correspondiente Distancia Mínima sobre $GF(2^m)$, abreviado MD_{2^m}, es NP-completo. Luego esta prueba se basa en una transformación polinomial de Decodificación de Máxima Verosimilitud a MD_{2^m}. Sin embargo, esto no prueba que Distancia Mínima sea NP-completo, ya que el posible conjunto de entradas a Distancia Mínima es un pequeño subconjunto del conjunto de posibles entradas a MD_{2^m}. Para ello, se construye una aplicación del código $\mathbb{C}\#$ sobre $GF(2^m)$ a un código binario $\mathbb{C}$, de tal forma que la distancia mínima de $\mathbb{C}\#$ puede determinarse a partir de la distancia mínima de $\mathbb{C}$. Dado que la longitud de $\mathbb{C}$ está acotada por la longitud de un polinomio de $\mathbb{C}\#$, y el mapeo en sí se puede lograr en tiempo polinomial, esto completa la prueba de la NP-completitud de Distancia Mínima.

Definición 18. Sea A_i, también denotada por $A_i(\mathcal{C})$, el número de palabras código con peso i en $\mathcal{C}$. Se dice que la lista A_i para $0 \leq i \leq n$ es la distribución del peso o espectro del peso de $\mathcal{C}$.

2.5 CLASIFICACIÓN POR ISOMETRÍA

Como hemos visto, las propiedades de un código dependen principalmente de las distancias de Hamming entre sus palabras y entre palabras codificadas y no codificadas. Además, puede ser que un código pueda relacionarse con otro por medio de una aplicación que conserve las distancias de Hamming. De esta forma, podemos definir una relación de equivalencia entre dos códigos que preservan la distancia de Hamming.

Sean C y C' dos $[n,k]_q$ códigos, se dice que son de la *misma cualidad* si existe una aplicación

$$\iota : F_q^n \to F_q^n$$

con $\iota(C) = C'$ que preserva la distancia de Hamming, es decir,

$$d(w,w') = d(\iota(w),\iota(w')), \qquad \forall w,w' \in H(n,q).$$

Las aplicaciones con la propiedad anterior se llaman *isometrías*.

Definición 19. Dos códigos lineales $C, C' \subseteq H(n,q)$ se llaman *isométricos* si existe una isometría de $H(n,q)$ que aplica C sobre C'.

Las permutaciones de las coordenadas son isometrías, que se denominan *isometrías permutacionales*.

Definición 20. Sea S_n el grupo isométrico en el conjunto $X = n = \{0, ..., n-1\}$. Dos códigos lineales $C, C' \subseteq H(n,q)$ son isométricos permutacionalmente si existe una isometría permutacional de $H(n,q)$ que aplica C sobre C'. Esto es, hay una permutación π en el grupo simétrico S_n tal que

$$C' = \pi(C) = \{\pi(c) : c \in C\}, \quad \text{and} \quad d(c,\tilde{c}) = d(\pi(c), \pi(\tilde{c})), \qquad \forall c, \tilde{c} \in C,$$

donde

$$\pi(c) = \pi(c_0, ..., c_{n-1}) := \left(c_{\pi^{-1}(0)}, ..., c_{\pi^{-1}(n-1)}\right)$$

2.6 ALGORITMO PARA EL CÁLCULO DE LA DISTANCIA

Como hemos visto, la distancia mínima es importante en un código lineal. Sin embargo, calcular este parámetro para un código dado puede resultar laborioso. A continuación presentaremos el algoritmo de BZ para el cálculo de la distancia, que tiene eficiencia exponencial [4, Sección 1.8]. Este algoritmo destaca por ser el algoritmo más rápido para calcular la distancia mínima de un código lineal. En particular, si el código es binario se considera eficaz.

Input: matriz generadora $G_1 = (I_k|A_1)$ de $\mathcal{C}$
Output: distancia mínima $\bar{d}_i$

$m \longleftarrow 2$
$k_1 \longleftarrow k$
while *rank* $(A_m) \neq 0$ **do**
  Aplicar la eliminación de Gauss y posibles permutaciones de las columnas de la matriz A_{m-1} desde $G_{m-1} = \left(\begin{array}{c|c|c} A'_{m-1} & \begin{array}{c} I_{k_{m-1}} \\ \hline 0 \end{array} & \begin{array}{c} A_{m-1} \\ \hline 0 \end{array} \end{array} \right)$ para obtener la matriz generadora $G_m = \left(\begin{array}{c|c|c} A'_m & \begin{array}{c} I_{k_m} \\ \hline 0 \end{array} & \begin{array}{c} A_m \\ \hline 0 \end{array} \end{array} \right)$
end
$C_0 \longleftarrow \{0\}$
$i \longleftarrow 0$
while $\bar{d}_i > \underline{d}_i$ **do**
  $i \longleftarrow i + 1$
  $C_i \longleftarrow C_{i-1} \cup \bigcup_{j=1}^{m} \left\{ v \cdot G_j : v \in \mathbb{F}_q^k,\ wt(v) = i \right\}$
  $\bar{d}_i \longleftarrow \min \{ wt(c) : c \in C_i,\ c \neq 0 \}$
  $\underline{d}_i \longleftarrow \sum_{\substack{j=1 \\ k-k_j \leq i}}^{m} \left((i+1) - (k - k_j) \right)$
end

Algoritmo 1: Algoritmo de Brouwer-Zimmermann: cálculo de la distancia mínima de un $[n,k]$ código lineal $\mathcal{C}$.

Veamos, en efecto, que este algoritmo determina la distancia mínima. Consideramos $\mathcal{C}$ un $[n,k]_q$ código lineal y G una matriz generadora. Como el código es lineal, existen matrices $M \in M_n(q)$ y $B \in GL_k(q)$ tales que $B \cdot G \cdot M^T$ es una matriz generadora en forma estándar. De hecho, usando el método de Gauss, podemos obtener una matriz generadora en forma estándar mediante operaciones elementales en las filas y permutaciones de las columnas. En este caso para realizar las operaciones elementales en las filas multiplicaremos desde la izquierda por una matriz $B_1 \in GL_k(q)$, y para permutar las columnas multiplicaremos desde la derecha por la traspuesta de una matriz de permutaciones M_{π_1}, de tal forma que

$$G_1 := B_1 \cdot G \cdot M_{\pi_1}^T = (I_{k_1}|A_1)\,,$$

donde $k_1 = k$.

Ahora que tenemos la matriz generadora en forma estándar, necesitaremos tranformarla de nuevo. Si A_1 no es una matriz nula ni vacía, su rango será k_2 con $0 < k_2 \leq k_1$. Aplicando la eliminación de Gauss, podemos obtener k_2 vectores unitarios diferentes en las $n - k_1$ columnas. Esto es, podemos multiplicar a G_1 desde la izquierda por una matriz $B_2 \in GL_k(q)$ y desde la derecha por la traspuesta de una matriz de permutaciones M_{π_2} con $\pi_2(j) = j$ para $0 \leq j < k_1$, obteniendo

$$G_2 := B_2 \cdot G_1 \cdot M_{\pi_2}^T = \left(\begin{array}{c|c|c} A_2' & \begin{array}{c} I_{k_2} \\ \hline 0 \end{array} & \begin{array}{c} A_2 \\ \hline 0 \end{array} \end{array} \right).$$

De esta forma, obtenemos que la matriz A_2' es una matriz $k \times k_1$ y la matriz A_2 es una matriz $k_2 \times (n - k_1 - k_2)$. Los ceros indican las matrices nulas.

Una vez que tenemos la matriz generadora de esta forma, realizaremos un procedimiento iterativo aplicando la eliminación de Gauss y posibles permutaciones de las columnas de la matriz A_2 hasta que su rango sea o. Esto es, sea $i \geq 2$, la matriz A_i tiene rango k_{i+1} con $0 < k_{i+1} \leq k_i$. En cada iteración, obtendremos matrices regulares $B_{i+1} \in GL_k(q)$, matrices de permutación $M_{\pi+1} \in M_n(q)$ con $\pi_{i+1}(j) = j$ para $0 \leq j < k_1 + \cdots + k_i$ y matrices generadoras

$$G_{i+1} := B_{i+1} \cdot G_i \cdot M_{\pi_{i+1}}^T = \left(\begin{array}{c|c|c} A_{i+1}' & \begin{array}{c} I_{k_{i+1}} \\ \hline 0 \end{array} & \begin{array}{c} A_{i+1} \\ \hline 0 \end{array} \end{array} \right),$$

donde A_{i+1}' es una matriz $k \times (k_1 + \cdots + k_i)$ y A_{i+1} es una matriz $k_{i+1} \times (n - k_1 - \cdots - k_{i+1})$. Repitiendo este procedimiento, finalmente obtendremos una matriz generadora G_m tal que

$$G_m = B_m \cdot G_{m-1} \cdot M_{\pi_m}^T = \left(\begin{array}{c|c|c} A_m' & \begin{array}{c} I_{k_m} \\ \hline 0 \end{array} & \begin{array}{c} A_m \\ \hline 0 \end{array} \end{array} \right),$$

donde A_m es una matriz vacía (que significa que no tiene columnas) o es una matriz nula. Entonces, $k_1 + \cdots + k_m \leq n$ y A_m tiene $n - k_1 - \cdots - k_m$ columnas. En consecuencia, la matriz generadora G_m tiene $n - k_1 - \cdots - k_m$ columnas nulas y por lo cual todos los elementos del código generado por G_m tiene peso menor que $k_1 + \cdots + k_m$.

Definimos para $1 \leq i \leq k$ los siguientes conjuntos:

$$C_i := \bigcup_{j=1}^{m} \left\{ v \cdot G_j : v \in \mathbb{F}_q^k,\ wt(v) \leq i \right\}$$

Claramente, estos conjuntos forman una cadena ascendente

$$C_1 \subseteq \cdots \subseteq C_k,$$

y por lo tanto los pesos mínimos

$$\bar{d}_i := \text{mín} \{ wt(c) : c \in C_i,\ c \neq 0 \}$$

forman una secuencia decreciente

$$\bar{d}_1 \geq \cdots \geq \bar{d}_k = dist(C).$$

En la mayoría de los casos, no será necesario calcular todos estos valores. En efecto, primero calcularemos $\bar{d}_1$. Después, si $\bar{d}_i$ ya ha sido calculada para algún $i \geq 1$, lo compararemos con $\underline{d}_i$, que es un límite inferior para el peso de los elementos en $C \setminus C_i$. Si $\bar{d}_i \leq \underline{d}_i$, hemos terminado. Si no, repetiremos el mismo proceso para $\bar{d}_{i+1}$.

Para calcular los límites inferiores para los pesos en los complementos $C \setminus C_i$, elegimos un elemento $c \in C \setminus C_i$. Como $c \notin C_i$, entonces existe, para cada j, un vector $v^{(j)} \in \mathbb{F}_q^k$ tal que

$$c = v^{(j)} \cdot G_j, \quad 1 \leq j \leq m, \text{ y } wt(v^{(j)}) \geq i+1.$$

Para estimar el peso de c, nos basaremos en los distintos lugares de información en G_i, es decir, las columnas que contienen la matriz identidad I_{k_j}. Estas son las columnas de índice r para $k_i + \cdots + k_{j-1} \leq r < k_1 + \cdots + k_j$. Nos interesan las k_j coordenadas c_r de $c = v^{(j)} \cdot G_j$ correspondientes a estas k_j columnas. Como $v^{(j)}$ tiene longitud k, estas entradas de c contribuyen al menos el valor de $i + 1 - (k - k_j)$ al peso de c. Además, como los conjuntos son disjuntos, para diferentes j, tenemos que

$$wt(c) \geq \sum_{j=1}^{m} \left(i + 1 - (k - k_j)\right).$$

Restringiendo esta suma a sumandos positivos, obtenemos el límite inferior

$$wt(c) \geq \sum_{j\,:\,k-k_j \leq i} \left(i + 1 - (k - k_j)\right) =: \underline{d}_i.$$

La secuencia de estos límites es incremental, pues el primer sumando es $i + 1$:

$$2 \leq \underline{d}_1 < \cdots < \underline{d}_k.$$

Además,

$$\underline{d}_k = m + \sum_{j=1}^{m} k_j.$$

Finalmente, como $wt(c) \leq k_1 + \cdots + k_m$ para todo $c \in C$, existe un índice i_0 tal que

$$\bar{d}_{i_0} \leq \underline{d}_{i_0}.$$

Para este i_0 se cumple que

$$\bar{d}_{i_0} := \text{mín}\left\{wt(c) : c \in C_{i_0},\ c \neq 0\right\},$$

y se sigue cumpliendo que

$$\underline{d}_{i_0} \leq \min\{wt(c) : c \in C \setminus C_{i_0}\}.$$

Por lo que

$$\bar{d}_{i_0} = dist(C),$$

y la palabra código de peso $dist(C)$ se encuentra en C_{i_0}.

Ejemplo 7. Consideramos el $[7,3]$ código binario $\mathcal{C}$ con matriz generadora

$$G_1 = \left(\begin{array}{ccc|cccc} 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{array}\right).$$

Aplicaremos el algoritmo de BZ para calcular la distancia mínima de este código.

Observamos que la matriz generadora G_1 ya está en forma estándar y su conjunto de información es $\{0,1,2\}$. Luego lo primero que haremos será aplicar el método de Gauss y permutaciones de las columnas para transformar esta matriz generadora en una matriz generadora de la forma

$$\left(\begin{array}{c|c|c} A_2' & I_{k_2} & A_2 \\ \hline & 0 & 0 \end{array}\right),$$

para alguna matriz A_2' de dimensión $3 \times k_1$ y alguna matriz A_2 de dimensión $k_2 \times (7 - k_1 - k_2)$. En nuestro caso, tenemos que

$$G_2 := \left(\begin{array}{ccc|ccc|c} 0 & 1 & 1 & 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 \end{array}\right).$$

Esta matriz generadora tiene conjunto de información $\{3,4,5\}$. El siguiente paso será realizar de nuevo la eliminación de Gauss y posibles permutaciones de las columnas de la matriz A_2 hasta que su rango sea 0. El algoritmo nos dará la siguiente matriz generadora

$$G_3 := \left(\begin{array}{cccccc|c} 0 & 1 & 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 \end{array}\right).$$

Esta matriz generadora tiene conjunto de información $\{6\}$. El conjunto C_1 está formado por las filas de las tres matrices generadoras G_1, G_2 y G_3. Ahora, calculamos el peso de cada uno de ellos, esto es, el número de coordenadas no nulas de cada elemento y obtenemos que todos tienen peso 4, por lo que $\bar{d}_1 = 4$. El límite inferior del peso mínimo de los vectores que no

pertenecen a C_1 es $\underline{d}_1 = 1 + 1 - (3 - 3) + 1 + 1 - (3 - 3) = 4$. Concluimos que $d = \bar{d}_1 = 4$ es la mínima distancia de C.

3

CÓDIGOS DE GOPPA

En [5], Goppa describió una nueva clase de códigos de corrección de errores lineales. Lo más importante es que algunos de estos códigos excedían el límite asintótico de Gilbert-Varshamov [6], una hazaña que muchos teóricos códigos pensaban que nunca podría lograrse. En este capítulo vamos a estudiar los códigos de Goppa [1, Sección 13.2.2] y sus propiedades más importantes, así como su codificación y decodificación [5].

3.1 CÓDIGOS CLÁSICOS DE GOPPA

Fijado el cuerpo de extensión $\mathbb{F}_{q^t}$ de $\mathbb{F}_q$, sea $L = \{\gamma_0, ..., \gamma_{n-1}\}$ una tupla de n elementos distintos de $\mathbb{F}_{q^t}$ y sea $g(x) \in \mathbb{F}_{q^t}[x]$ con $g(\gamma_i) \neq 0$ para $0 \leq i \leq n-1$. Entonces el *código de Goppa* $\Gamma(L, g)$ es el conjunto de vectores $c_0 \cdots c_{n-1} \in \mathbb{F}_q^n$ tal que

$$\sum_{i=0}^{n-1} \frac{c_i}{x - \gamma_i} \equiv 0 \pmod{g(x)}. \tag{3}$$

De esta forma, cuando la parte de la izquierda está escrita como una función racional, significa que el numerador es un múltiplo de $g(x)$. Además, trabajar módulo $g(x)$ es como trabajar en el anillo $\mathbb{F}_{q^t}[x]/(g(x))$, y la hipótesis $g(\gamma_i) \neq 0$ garantiza que $x - \gamma_i$ es invertible en este anillo. Se llama a $g(x)$ el *polinomio de Goppa* de $\Gamma(L, g)$. Notemos que el código BCH de longitud n y la distancia elegida δ es el código de Goppa $\Gamma(L, g)$ con $L = \{1, \beta^{-1}, ..., \beta^{1-n}\}$ y $g(x) = x^{\delta-1}$.

A continuación, buscaremos una matriz de paridad para $\Gamma(L, g)$. Para ello, observamos que

$$\frac{1}{x - \gamma_i} \equiv -\frac{1}{g(\gamma_i)} \frac{g(x) - g(\gamma_i)}{x - \gamma_i} \pmod{g(x)}$$

ya que, comparando numeradores, $1 \equiv -g(\gamma_i)^{-1} (g(x) - g(\gamma_i)) \pmod{g(x)}$. Así que por (3) $\mathbf{c} = c_0 \cdots c_{n-1} \in \Gamma(L, g)$ si y solo si

$$\sum_{i=0}^{n-1} c_i \frac{g(x) - g(\gamma_i)}{x - \gamma_i} g(\gamma_i)^{-1} \equiv 0 \pmod{g(x)}. \tag{4}$$

Supongamos que $g(x) = \sum_{j=0}^{w} g_j x^j$ con $g_j \in \mathbb{F}_{q^t}$, donde $w = \mathrm{gr}(g(x))$. Entonces

$$\frac{g(x)-g(\gamma_i)}{x-\gamma_i}g(\gamma_i)^{-1} = g(\gamma_i)^{-1}\sum_{j=1}^{w} g_j \sum_{k=0}^{j-1} x^k \gamma_i^{j-1-k} = g(\gamma_i)^{-1}\sum_{k=0}^{w-1} x^k \left(\sum_{j=k+1}^{w} g_j \gamma_i^{j-1-k}\right)$$

Por lo tanto, por (4), estableciendo los coeficientes de x^k iguales a 0, en el orden $k = w-1, w-2, ..., 0$, tenemos que $\mathbf{c} \in \Gamma(L,g)$ si y solo si $Hc^T = 0$, donde

$$H = \begin{pmatrix} h_0 g_w & \cdots & h_{n-1} g_w \\ h_0(g_{w-1} + g_w\gamma_0) & \cdots & h_{n-1}(g_{w-1} + g_w\gamma_{n-1}) \\ & \vdots & \\ h_0 \sum_{j=1}^{w}\left(g_j + \gamma_0^{j-1}\right) & \cdots & h_{n-1}\sum_{j=1}^{w}\left(g_j + \gamma_{n-1}^{j-1}\right) \end{pmatrix}, \tag{5}$$

con $h_i = g(\gamma_i)^{-1}$.

Proposición 10. *La matriz H se puede reducir a una matriz H' de dimensión $w \times n$, donde*

$$H' = \begin{pmatrix} g(\gamma_0)^{-1} & \cdots & g(\gamma_{n-1})^{-1} \\ g(\gamma_0)^{-1}\gamma_0 & \cdots & g(\gamma_{n-1})^{-1}\gamma_{n-1} \\ & \vdots & \\ g(\gamma_0)^{-1}\gamma_0^{w-1} & \cdots & g(\gamma_{n-1})^{-1}\gamma_{n-1}^{w-1} \end{pmatrix}. \tag{6}$$

Las entradas de H' están en $\mathbb{F}_{q^t}$. Eligiendo una base de $\mathbb{F}_{q^t}$ sobre $\mathbb{F}_q$, cada elemento de $\mathbb{F}_{q^t}$ se puede representar como un vector columna $t \times 1$ sobre $\mathbb{F}_q$. Reemplazando cada entrada de H' por su correspondiente vector columna, obtenemos una matriz H'' de dimensión $tw \times n$ sobre $\mathbb{F}_q$ que tiene la propiedad de que $\mathbf{c} \in \mathbb{F}_q^n$ está en $\Gamma(L,g)$ si y solo si $H''c^T = 0$.

El siguiente resultado nos muestra los límites en la dimensión y la distancia mínima de un código de Goppa.

Teorema 9. *Con la notación de esta sección, sea $\Gamma(L,g)$ un código de Goppa tal que $\mathrm{gr}(g(x)) = w$ entonces es un $[n,k,d]$ código con $k \geq n - wt$ y $d \geq w+1$.*

Demostración. Las filas de H'' pueden ser dependientes, luego esta matriz tiene rango como máximo wt. Por lo que $\Gamma(L,g)$ tiene dimensión al menos $n - wt$. Si una palabra código $\mathbf{c} \in \Gamma(L,g)$ tiene peso w o menos, entonces el lado izquierdo de 3 es una función racional, donde el numerador tiene grado $w-1$ o menos; pero este numerador tiene que ser múltiplo de $g(x)$, lo cual es una contradicción pues el grado de g es w. □

Corolario 1. *Si $\Gamma(L,g)$ es un código de Goppa tal que $\mathrm{gr}(g(x)) = w$, entonces puede corregir hasta*

$$\left\lfloor \frac{w}{2} \right\rfloor$$

errores.

Demostración. El teorema 7 afirma que es posible corregir hasta

$$\left\lfloor \frac{d(\Gamma(L,g))-1}{2} \right\rfloor$$

errores. Además, por el teorema 9 sabemos que la distancia mínima de un código de Goppa $\Gamma(L,g)$ tal que $\mathrm{gr}(g(x)) = w$ es $d(\Gamma(L,g)) = w+1$. Por lo que

$$\left\lfloor \frac{d(\Gamma(L,g))-1}{2} \right\rfloor = \left\lfloor \frac{(w+1)-1}{2} \right\rfloor = \left\lfloor \frac{w}{2} \right\rfloor.$$

□

3.1.1 *Códigos binarios de Goppa*

Los códigos binarios de Goppa son códigos de corrección de errores que pertenecen a la clase de los códigos de Goppa que acabamos de estudiar. La estructura binaria le da más ventajas matemáticas sobre variantes no binarias y, además, tienen propiedades interesantes adecuadas para la construcción del criptosistema McEliece.

Definición 21. Fijada la extensión de cuerpos $\mathbb{F}_{2^m}$ de $\mathbb{F}_2$, sea $L = \{\gamma_0, ..., \gamma_{n-1}\} \in \mathbb{F}_{2^m}^n$ una tupla de n elementos distintos de $\mathbb{F}_{2^m}$ y sea $g(x) \in \mathbb{F}_{2^m}[x]$ con $g(\gamma_i) \neq 0$ para $0 \leq i \leq n-1$. Entonces el *código binario de Goppa* $\Gamma(L,g)$ es el conjunto de vectores $c_0 \cdots c_{n-1} \in \{0,1\}^n$ tal que

$$\sum_{i=0}^{n-1} \frac{c_i}{x-\gamma_i} \equiv 0 \pmod{g(x)} \tag{7}$$

Observemos que si $g(x)$ es un polinomio irreducible, todos los elementos $\gamma \in \mathbb{F}_{2^m}$ satisfacen $g(\gamma) \neq 0$. A los códigos que cumplan esta propiedad los llamaremos *códigos binarios de Goppa irreducibles*.

La importancia de estos códigos se debe a que pueden doblar la capacidad correctora de los códigos generales de Goppa.

3.1.2 *Codificación de los códigos de Goppa*

3.1.3 *Decodificación de los códigos de Goppa*

Como hemos visto, al transmitir una palabra código a un receptor, éste podría recibir la palabra alterada. Para que el receptor pueda determinar el mensaje original, necesitaremos decodificar el mensaje recibido, [5]. Sea $\Gamma(L,g)$ un código de Goppa, donde $L = \{\gamma_0, ..., \gamma_{n-1}\}$ es una tupla de n elementos distintos de $\mathbb{F}_{q^t}$. Supongamos que e es el vector de errores que se añade a la palabra código c transmitida, entonces la palabra recibida y está dada por

$$y = c + e,$$

de donde

$$\sum_{\gamma \in L} \frac{y_\gamma}{x - \gamma} = \sum_{\gamma \in L} \frac{c_\gamma}{x - \gamma} + \sum_{\gamma \in L} \frac{e_\gamma}{x - \gamma}.$$

Como c es una palabra código, la primera sumatoria de la parte derecha desaparece al aplicar el módulo $g(x)$, y tenemos que

$$\sum_{\gamma \in L} \frac{y_\gamma}{x - \gamma} = \sum_{\gamma \in L} \frac{e_\gamma}{x - \gamma} \pmod{g(x)}.$$

Definiremos su síndrome como el polinomio $S(x)$ de grado menos que $\mathrm{gr}(g(x))$ tal que

$$S(x) = \sum_{\gamma \in L} \frac{y_\gamma}{x - \gamma} \pmod{g(x)}.$$

Acabamos de ver que

$$S(x) = \sum_{\gamma \in L} \frac{e_\gamma}{x - \gamma} \pmod{g(x)}.$$

Sea M un subconjunto de L tal que $e_\gamma \neq 0$ si y solo si $\gamma \in M$. Entonces

$$S(x) = \sum_{\gamma \in M} \frac{e_\gamma}{x - \gamma} \pmod{g(x)}. \tag{8}$$

De esta forma, ahora podemos introducir el polinomio cuyas raíces son las ubicaciones de los errores,

$$\sigma(x) = \prod_{\gamma \in M} (x - \gamma). \tag{9}$$

Sin embargo, para los códigos de Goppa es más conveniente definir una variante de este polinomio de la siguiente forma.

$$\eta(x) = \sum_{\gamma \in M} e_\gamma \prod_{\partial \in M \setminus \{\gamma\}} (x - \partial) \tag{10}$$

Observemos que de esta forma $\sigma(x)$ y $\eta(x)$ deben ser primos relativos.

Derivando la expresión de $\sigma(x)$, tenemos que

$$\sigma'(x) = \sum_{\gamma \in M} \prod_{\partial \in M \setminus \{\gamma\}} (x - \partial), \tag{11}$$

de donde, para cada $\gamma \in M$,

$$\eta(\gamma) = e_\gamma \prod_{\partial \in M \setminus \{\gamma\}} (\gamma - \partial) = e_\gamma \sigma'(\gamma),$$

por lo que $e_\gamma = \frac{\eta(\gamma)}{\sigma'(\gamma)}$. De esta forma, una vez que hemos calculado los polinomios σ y η, las coordenadas del vector error vienen dadas por

$$e_\gamma = \begin{cases} 0 & \text{si} \quad \sigma(\gamma) \neq 0 \\ \\ \frac{\eta(\gamma)}{\sigma'(\gamma)} & \text{si} \quad \sigma(\gamma) = 0 \end{cases}$$

donde $\sigma'(x)$ es la derivada de $\sigma(x)$.

Lo esencial para decodificar los códigos de Goppa es determinar los coeficientes de los polinomios σ y η. Para ello, tenemos que relacionar σ y η al síndrome de la ecuación 8. Esto se consigue multiplicando las ecuaciones 8 y 9, obteniendo

$$S(x) \cdot \sigma(x) \equiv \eta(x) \pmod{g(x)}. \tag{12}$$

La ecuación 12 es la *ecuación clave* para decodificar los códigos de Goppa. Dado $g(x)$ y $S(x)$, el problema de decodificar consiste en encontrar polinomios de grado bajo $\sigma(x)$ y $\eta(x)$ que satisfacen 12.

Reduciendo cada potencia de $x \pmod{g(x)}$ e igualando coeficientes de $1, x, ..., x^{\mathrm{gr}(g)-1}$, tenemos que 12 es un sistema de $\mathrm{gr}(G)$ ecuaciones lineales donde las incógnitas son los coeficientes de σ y η. Por lo tanto, para probar que el decodificador es capaz de corregir todos los patrones hasta t errores, basta con probar que 12 tiene una única solución con grados de σ y de η suficientemente pequeños. Esto equivale a que el conjunto de ecuaciones lineales correspondientes sean linealmente independientes.

Supongamos que existen dos pares diferentes de soluciones a 12:

$$S(x)\sigma^{(1)}(x) \equiv \eta^{(1)}(x) \pmod{g(x)}, \tag{13}$$

$$S(x)\sigma^{(2)}(x) \equiv \eta^{(2)}(x) \pmod{g(x)}, \tag{14}$$

donde $\sigma^{(1)}(x)$ y $\eta^{(1)}(x)$ son primos relativos, al igual que $\sigma^{(2)}(x)$ y $\eta^{(2)}(x)$. Además, $\sigma^{(1)}(x)$ y $g(x)$ no pueden tener ningún factor en común, pues en ese caso ese factor podría dividir a $\eta^{(1)}(x)$, contradiciendo que $\sigma^{(1)}(x)$ y $\eta^{(1)}(x)$ son primos relativos. Así, podemos dividir 13 por $\sigma^{(1)}(x)$ y obtenemos

$$S(x) \equiv \frac{\eta^{(1)}(x)}{\sigma^{(1)}(x)} \pmod{g(x)}.$$

De la misma forma para 14,

$$S(x) \equiv \frac{\eta^{(2)}(x)}{\sigma^{(2)}(x)} \pmod{g(x)}$$

de donde,

$$\sigma^{(1)}(x)\eta^{(2)}(x) \equiv \sigma^{(2)}(x)\eta^{(1)}(x) \pmod{g(x)}. \tag{15}$$

Si $\mathrm{gr}(G) = 2t$ y $\mathrm{gr}(\sigma^{(1)}) \leq t$, $\mathrm{gr}(\sigma^{(2)}) \leq t$, $\mathrm{gr}(\eta^{(2)}) < t$ y $\mathrm{gr}(\eta^{(1)}) < t$, entonces se da la siguiente igualdad

$$\sigma^{(1)}(x)\eta^{(2)}(x) = \sigma^{(2)}(x)\eta^{(1)}(x). \tag{16}$$

Así, $\sigma^{(1)}$ divide a $\sigma^{(2)}\eta^{(1)}$, y como $\sigma^{(1)}$ y $\eta^{(1)}$ son primos relativos, $\sigma^{(1)}$ tiene que dividir a $\sigma^{(2)}$. Análogamente, $\sigma^{(2)}$ tiene que dividir a $\sigma^{(1)}$. Como ambos son mónicos, se tiene que $\sigma^{(1)} = \sigma^{(2)}$ y así, $\eta^{(1)} = \eta^{(2)}$. Con esto hemos probado que si el grado de G es $2t$, entonces 12 tiene una única solución cuando $\mathrm{gr}(\eta) < \mathrm{gr}(\sigma) \leq t$, luego el correspondiente sistema de ecuaciones lineales donde las incógnitas son los coeficientes de σ y η tiene que ser no singular. En el siguiente teorema se concluye este resultado.

Teorema 10. *Si* $\mathrm{gr}(g(x)) = 2t$, *entonces hay un algoritmo de decodificación algebraica de corrección de t errores para el código q-ario de Goppa con el polinomio de Goppa* $g(x)$.

Estudiemos ahora este resultado en el caso binario, primero observamos que ya que todos los e_γ distintos de cero son iguales a 1, entonces 10 y 11 coincidan. De esta forma, la ecuación 15 ahora pasa a ser

$$\sigma^{(1)}\left(\sigma^{(2)}\right)' \equiv \sigma^{(2)}\left(\sigma^{(1)}\right)' \pmod{g(x)}$$

Ahora, cuando σ sea par escribiremos en su lugar $\hat{\sigma}$, mientras que cuando σ sea impar escribiremos en su lugar $x\sigma'$. Así, tenemos que

$$\begin{aligned}\left(\hat{\sigma}^{(1)} + x\sigma^{(1)'}\right)\sigma^{(2)'} &\equiv \left(\hat{\sigma}^{(2)} + x\sigma^{(2)'}\right)\sigma^{(1)'} \\ \hat{\sigma}^{(1)}\sigma^{(2)'} + \hat{\sigma}^{(2)}\sigma^{(1)'} &\equiv 0 \pmod{g(x)}.\end{aligned}$$

El lado izquierdo es un cuadrado perfecto, pues todos los polinomios de ese lado son pares. Esto implica que

$$\hat{\sigma}^{(1)}\sigma^{(2)'} + \hat{\sigma}^{(2)}\sigma^{(1)'} \equiv 0 \pmod{\bar{G}(x)}$$

donde $\bar{G}(x)$ es múltiplo de $g(x)$ de menor grado ya que $\bar{G}$ es un cuadrado perfecto. Por lo que, si $\mathrm{gr}(\bar{G}) = 2t$, $\mathrm{gr}(\sigma^{(1)}) \leq t$ y $\mathrm{gr}(\sigma^{(2)}) \leq t$, entonces

$$\hat{\sigma}^{(1)} \left(\sigma^{(2)}\right)' = \hat{\sigma}^{(2)} \sigma^{(1)'}.$$

Por la primalidad relativa, $\sigma^{(1)} = \sigma^{(2)}$. En el siguiente teorema se concluye este resultado.

Teorema 11. *Si* $\mathrm{gr}(g(x)) = t$ *y si* $g(x)$ *no tiene factores irreducibles repetidos, entonces hay un algoritmo de decodificación algebraica de corrección de* t *errores para el código binario de Goppa con el polinomio de Goppa* $g(x)$.

3.1.3.1 *Algoritmo de decodificación de Sugiyama*

El algoritmo de Sugiyama, descrito en [7], es una aplicación simple del algoritmo de Euclides para determinar el polinomio localizador de errores de una manera más eficiente.

Dado $g(x)$ un polinomio de Goppa con grado $2t$, el algoritmo de decodificación de Sugiyama para los códigos Goppa es el siguiente.

I. Calcular el síndrome $S(x)$.
II. Sean $r_{-1}(x) = g(x)$, $r_0(x) = S(x)$, $U_{-1}(x) = 0$ y $U_0(x) = 1$.
III. Buscar $q_i(x)$ y $r_i(x)$ aplicando el algoritmo de Euclides para encontrar el máximo común divisor de $r_{-1}(x)$ y $r_0(x)$ para $i = 1, ..., k$, hasta que k cumpla que $\mathrm{gr}(r_{k-1}(x)) \geq t$ y $\mathrm{gr}(r_k(x)) < t$:

$$r_{i-2}(x) = r_{i-1}(x) q_i(x) + r_i(x), \qquad \mathrm{gr}(r_i(x)) < \mathrm{gr}(r_{i-1})(x)$$

IV. Calcular $U_k(x)$, donde

$$U_i(x) = q_i(x) U_{i-1}(x) + U_{i-2}(x)$$

V. La solución viene dada por:

$$\eta(x) = (-1)^k \delta r_k(x)$$
$$\sigma(x) = \delta U_k(x)$$

4

IMPLEMENTACIÓN EN SAGEMATH DEL ALGORITMO DE CODIFICACIÓN Y DECODIFICACIÓN DE CÓDIGOS DE GOPPA

4.1 REPRESENTACIÓN DE CÓDIGOS DE GOPPA

```
    from sage.coding.linear_code import AbstractLinearCode
    from sage.coding.encoder import Encoder
    from sage.coding.decoder import Decoder

    class Goppa(AbstractLinearCode):
        r"""
        Implementation of Goppa codes.

        INPUT:
        - ``generating_pol`` -- a monic polynomial with coefficients in
        a finite fiel `\GF{p^m}`

        - ``defining_set`` -- tuple of n distinct elements of `\GF{p^m}`
        that are roots of `generating_pol
        """
        def __init__(self, defining_set, generating_pol):
            """
            Initialize.
            """
            if not generating_pol.is_monic():
                raise ValueError("ERROR. Generating polynomial isn't monic")

            for gamma in defining_set:
                if generating_pol(gamma) == 0:
                    raise ValueError("ERROR. Defining elements are roots of
                        generating polynomial")

            self._field = generating_pol.base_ring().prime_subfield()
            self._polynomial_ring = PolynomialRing(self._field, defining_set[0].
                parent().gen())
```

```
29
30              if (not self._field.is_field() or not self._field.is_finite()):
31                  raise ValueError("ERROR. Generating polynomial isn't definied
                        over a finite field")
32
33              self._length = len(defining_set)
34              self._generating_pol = generating_pol
35              self._defining_set = defining_set
36
37              super(Goppa, self).__init__(self._field, self._length, "GoppaEncoder"
                    , "GoppaDecoder")
38
39          def get_generating_pol(self):
40              """
41              Return the generating polynomial
42              """
43              return self._generating_pol
44
45          def get_defining_set(self):
46              """
47              Return the defining set
48              """
49              return self._defining_set
50
51          def get_parity_pol(self):
52              """
53              Return the parity polynomial
54              """
55              parity_pol = list()
56
57              for elem in self._defining_set:
58                  parity_pol.append((self._generating_pol.parent().gen() - elem).
                        inverse_mod(self._generating_pol))
59
60              return parity_pol
61
62          def parity_check_matrix(self):
63              """
64              Return a parity check matrix of the code.
65              """
66              # create a matrix with the coefficients of the parity polynomial as
                    columns
67              h = self.get_parity_pol()
68              H_aux = matrix(h[0].list())
69
70              for i in range(1, len(h)):
```

```
            aux = matrix(h[i].list())  # get coefficients of h[i]
            H_aux = H_aux.stack(aux)   # add at the end

        H_aux = H_aux.transpose()

        # calculate the maximum degree
        max_degree = 0

        for hi in H_aux:
            for hij in hi:
                if self._polynomial_ring(hij).degree() > max_degree:
                    max_degree = self._polynomial_ring(hij).degree()

        # extend each component to the base field
        H = Matrix(vector([0] * H_aux.ncols()))

        for hi in H_aux:
            matriz = Matrix(vector([0] * (max_degree + 1)).column())

            for hij in hi:
                aux = self._polynomial_ring(hij).list()

                while len(aux) != max_degree + 1:
                    aux.append(0)

                matriz = matriz.augment(vector(ZZ, aux).column())

            matriz = matriz.delete_columns([0])
            H = H.stack(matriz)

        H = H.delete_rows([0])

        return H

    def minimum_distance(self):
        """
        Return the minimum distance of the code.
        """
        return 2 * (self._generating_pol).degree() + 1

    def dimension(self):
        """
        Return the dimension of the code
        """
        H = self.parity_check_matrix()

```

```
            return self._length - rank(H)

        def _repr_(self):
            """
            Representation of a Goppa code
            """
            return "[{}, {}] Goppa code".format(self._length, self.dimension())
```

4.2 CODIFICACIÓN DE CÓDIGOS DE GOPPA

```
class GoppaEncoder(Encoder):
class GoppaEncoder(Encoder):
    r"""
    Encoder for Goppa codes

    INPUT:
    - ``code`` -- code associated with the encoder
    """
    def __init__(self, code):
        """
        Initialize.
        """
        super(GoppaEncoder, self).__init__(code)
        H = self.code().parity_check_matrix()
        self._G = Matrix(self.code()._polynomial_ring, transpose(H).left_kernel()
            .basis_matrix())

    def _repr_(self):
        """
        Representation of a encoder for a Goppa code
        """
        return "Encoder for {}".format(self.code())

    def get_generator_matrix(self):
        """
        Return a generador matrix of the code
        """
        return self._G

    def encode (self, m):
        """
        Return a codeword

        INPUT:
```

```
        - ``m``: a vector to encode
        """
        return m * self._G

    # TODO: add a method to add errors?

Goppa._registered_encoders["GoppaEncoder"] = GoppaEncoder
```

4.3 DECODIFICACIÓN DE CÓDIGOS DE GOPPA

```
    class GoppaDecoder(Decoder):
    r"""
    Decoder for Goppa codes

    INPUT:
    - ``code``: code associated with the decoder
    """

    def __init__(self, code):
        """
        Initialize
        """
        super(GoppaDecoder, self).__init__(code, code.ambient_space(), "
            GoppaDecoder")

        self._generating_pol = self.code().get_generating_pol()
        self._defining_set = self.code().get_defining_set()

    def get_syndrome(self, c):
        """
        Return the syndrome polynomial

        INPUT:
        - ``c``: a codeword of ``self``
        """
        F = self.code()._field
        L = self._defining_set[0].parent()

        embFL = FiniteFieldHomomorphism_generic(Hom(F,L))
        secLF = embFL.section()

        h = self.code().get_parity_pol()

        syndrome = 0
```

```
34
35          for i in range(len(h)):
36              syndrome = syndrome + embFL(c[i])*h[i]
37
38          return syndrome
39
40      def get_generating_pol(self):
41          """
42          Return the generating polynomial
43          """
44          return self._generating_pol
45
46      def decode_to_code(self, word):
47          r"""
48          Corrects the errors in ``word`` and returns a codeword.
49          INPUT:
50          - ``word`` -- a codeword of ``self``
51          """
52          i = 1
53
54          # Step 1
55          S = self.get_syndrome(word)
56
57          # Step 2
58          r_prev = self.get_generating_pol()
59          t = self.get_generating_pol().degree()/2
60          r_i = S
61          U_prev = 0
62          U_i = 1
63
64          # Step 3 y 4
65          while r_i.degree() >= t:
66              (d, u, v) = xgcd(r_prev, r_i)
67              q_i = -v//u
68              r_i = d//u
69              U_i = q_i * U_i + U_prev
70              r_prev = r_i
71              U_prev = U_i
72              i += 1
73
74          # Step 5
75          # make U_i monic
76          coeffs_U_i = U_i.list()
77          n = len(coeffs_U_i)
78          delta = 1
79
```

```
        if coeffs_U_i[n-1] != 1:
            delta = 1/coeffs_U_i[n-1]

        eta = (-1)^i * delta * r_i
        sigma = U_i * delta

        # roots of sigma are the locations of the errors
        roots = []

        for root in sigma.roots():
            roots.append(self._defining_set.index(root[0]))

        error = [0] * len(self._defining_set)

        for root in roots:
            error[root] = eta

        x = word - vector(self.code()._field, error)

        return x

Goppa._registered_decoders["GoppaDecoder"] = GoppaDecoder
```

4.4 EJEMPLOS

Sea el cuerpo de extensión $\mathbb{F}_{2^3}$ de $\mathbb{F}_2$ y sea $g(x) = x^2 + x + 1 \in \mathbb{F}_{2^3}[x]$. Definimos L como una tupla de n elementos distintos de $\mathbb{F}_{q^t}$ tales que sus elementos no son raíces de g.

El código de Goppa asociado al conjunto de definición L y al polinomio g es un $[8, 2]$ código de Goppa. Este código tiene distancia mínima 5 y su matriz de paridad es la siguiente:

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

El codificador asociado al $[8, 2]$ código de Goppa tiene la siguiente matriz generadora:

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

Sea $(0,1) \in \mathbb{F}_2$ una palabra, la podemos codificar con la matriz anterior:

$$x = (0,1,1,1,1,1,1,0).$$

Ahora, queremos enviar la palabra codificada x a nuestro destinatario, pero para ello le añadimos un error en una posición aleatoria. Sea e el error definido como:

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

La palabra que enviaremos será:

$$y = x + e = (0,1,1,1,0,1,1,0).$$

El destinatario decodificará esta palabra con el algoritmo de Sugiyama obteniendo:

$$(0,1,1,1,1,1,1,0),$$

que coincide con x.

5

CRIPTOGRAFÍA POST-CUÁNTICA BASADA EN CÓDIGOS

El principal objetivo de la criptografía es proporcionar confidencialidad mediante métodos de cifrado. Cuando queremos enviar un mensaje a un destinatario, el canal de comunicación puede ser inseguro y otras personas podrían leerlo o incluso modificarlo de tal forma que el destinatario no se diera cuenta. Para prevenir estos ataques nos será de utilidad la criptografía.

5.1 INTRODUCCIÓN

En general, los métodos de cifrado consisten en encriptar el mensaje, *texto plano*, antes de ser transmitido, de esta forma obtenemos un *texto cifrado* o *criptograma*. Este texto cifrado se transmite al destinatario, quien lo *desencripta* mediante una *clave de descifrado*, la cual solo conocen el receptor y el emisor y previamente la intercambiaron.

Formalmente, dado un conjunto de los mensajes (textos planos) $\mathcal{M}$, un conjunto de los criptogramas $\mathcal{C}$ y el espacio de claves $\mathcal{K}_p \times \mathcal{K}_s$, un *criptosistema* viene definido por dos aplicaciones

$$E : \mathcal{K}_p \times \mathcal{M} \to \mathcal{C},$$
$$D : \mathcal{K}_s \times \mathcal{C} \to \mathcal{M},$$

tales que para cualquier clave $k_p \in \mathcal{K}_p$, existe una clave $k_s \in \mathcal{K}_s$ de manera que dado cualquier mensaje $m \in \mathcal{M}$,

$$D(k_s, E(k_p, m)) = m. \tag{17}$$

Para simplificar la notación de las funciones de cifrado y descifrado usaremos, fijadas las claves $k_p \in \mathcal{K}_p$ y su correspondiente $k_s \in \mathcal{K}_s$:

$$E_{k_p} : \mathcal{M} \to \mathcal{C}, \ \left[E_{k_p}(m) = E(k_p, m)\right]$$
$$D_{k_s} : \mathcal{C} \to \mathcal{M}, \ [D_{k_s}(c) = D(k_s, c)]$$

La propiedad 17 se transforma en

$$D_{k_s}\left(E_{k_p}(m)\right) = m.$$

Para encriptar y desencriptar existen *algoritmos de cifrado* y *de descifrado*, respectivamente, y cada uno usará una clave secreta. Si esta clave es la misma en ambos algoritmos, diremos que los métodos de encriptación son *simétricos*. Algunos ejemplos importantes de estos métodos son DES (*Data Encryption Standard*) y AES (*Advanced Encryption Standard*).

En 1976, Diffie y Hellman introdujeron un concepto revolucionario, la *criptografía de Clave Pública*, también llamada *criptografía asimétrica*, que permitió dar una solución al antiguo problema del intercambio de claves e indicar el camino a la firma digital. Los métodos de cifrado de *clave pública* son *asimétricos*. Cada receptor tiene una clave personal $k = (k_p, k_s)$, que consiste en dos partes: k_p es la clave de cifrado y es pública, y k_s es la clave de descifrado, que es privada. De esta forma, si queremos enviar un mensaje, lo encriptaremos mediante la clave pública k_p del receptor. Así, el receptor poddrá descifrar el texto cifrado usando su clave privada k_s, que solo la conoce él. Al ser la clave pública, cualquiera puede encriptar un mensaje usándola, sin embargo descifrarlo sin saber la clave privada será casi imposible.

5.2 OBJETIVOS DE LA CRIPTOGRAFÍA

Además de proporcionar confidencialidad, la criptografía proporciona soluciones para otros problemas:

1. *Confidencialidad*. La información solo puede ser accesible por las entidades autorizadas.
2. *Integridad de datos*. El receptor de un mensaje debería ser capaz de determinar que el mensaje no ha sido modificado durante la transmisión.
3. *Autenticidad*. El receptor de un mensaje debería ser capaz de verificar su origen.
4. *No repudio*. El emisor de un mensaje debería ser incapaz de negar posteriormente que envió el mensaje.

Para garantizar la integridad de los datos, hay métodos simétricos y de clave pública. El mensaje m es aumentado por un *código de autenticación de mensaje* (MAC). Este código es generado por un algoritmo que depende de la clave secreta. Así, el mensaje aumentado $(m, MAC(k, m))$ está protegido contra modificaciones. El receptor ahora puede comprobar la integridad del mensaje $(m, \bar{m})$ verificando que $MAC(k, m) = \bar{m}$. Más adelante veremos que la autenticación de códigos se puede implementar con funciones hash con clave.

Las firmas digitales requieren métodos de clave pública y proporcionan autenticación y no repudiablidad. Cualquier persona puede verificar si una firma digital es válida con la clave pública del firmante. Esto es, si firmamos con nuestra clave privada k, obtenemos la firma $Sign(k_s, m)$. El receptor recibe la firma s del mensaje m y comprueba con el algoritmo de verificación *Verify* que se cumple que $Verify(k_p, s, m) = ok$, siendo k_p la clave pública del emisor.

5.3 CRIPTOGRAFÍA SIMÉTRICA

Un criptosistema *simétrico*, como hemos visto, viene determinado por dos aplicaciones

$$E : \mathcal{K} \times \mathcal{M} \to \mathcal{C},$$
$$D : \mathcal{K} \times \mathcal{C} \to \mathcal{M},$$

tales que para cualquier clave $k \in \mathcal{K}$ y cualquier mensaje $m \in \mathcal{M}$,

$$D(k, E(k, m)) = m. \tag{18}$$

Fijada $k \in \mathcal{K}$, usaremos la notación

$$E_k : \mathcal{M} \to \mathcal{C},$$
$$D_k : \mathcal{C} \to \mathcal{M},$$

para las funciones de cifrado y descifrado. La propiedad 18 se transforma en

$$D_k\left(E_k(m)\right) = m.$$

Observamos que la clave que se usa para cifrar es la misma que se usa para descifrar.

Algunos ejemplos importantes de sistemas simétricos son DES (*Data Encryption Standard*) y AES (*Advanced Encryption Standard*).

5.4 CRIPTOGRAFÍA ASIMÉTRICA

A diferencia de la criptografía simétrica, en la criptografía asimétrica los participantes en la comunicación no comparten una clave secreta. Cada uno tiene un par de claves: la *clave secreta* k_s conocida solo por él y una *clave pública* conocida por todos.

Supongamos que Bob tiene un par de claves (k_p, k_s) y Alice quiere encriptar un mensaje m para Bob. Alice, como cualquier otra persona, conoce la clave pública k_p de Bob. Alice usa una función de encriptación E con la clave pública k_p de Bob para obtener el texto cifrado $c = E_{k_p}(m)$. Esto solo puede ser seguro si es prácticamente inviable calcular m de $c = E_{k_p}(m)$. Sin embargo, Bob sí es capaz de calcular el mensaje m, ya que puede usar su clave secreta. La función de encriptación E_{k_p} debe tener la propiedad de que su pre-imagen m del texto cifrado $c = E_{k_p}(m)$ sea fácil de calcular usando la clave secreta k_s de Bob, quien es el único que puede descifrar el mensaje encriptado.

En la criptografía de clave pública, necesitamos unas funciones $\left(E_{k_p}\right)_{k_p \in K_P}$ tales que cada función E_{k_p} se pueda cacular con un algoritmo eficiente. Sin embargo, su pre-imagen debería

ser prácticamente inviable de calcular. Estas familias $\left(E_{k_p}\right)_{k_p \in K_P}$ se denominan *funciones de una sola dirección*. En cada función E_{k_p} de la familia, tiene que haber una información secreta k_s para que el algoritmo sea eficiente y calcule la inversa de E_{k_p}. Las funciones con esa propiedad se denominan *funciones con trampilla*.

En 1976, Diffie y Hellman presentaron la idea de la criptografía de clave pública, es decir, introdujeron métodos de clave pública para el acuerdo de clave y, además, describieron cómo las firmas digitales funcionarían. El primer criptosistema de clave pública que podía servir como un mecanismo de acuerdo de clave y como una firma digital fue el criptosistema RSA, que actualmente es el más conocido y usado. Este criptosistema lleva el nombre de sus inventores: Rivest, Shamir y Adleman. El criptosistema RSA se basa en la dificultad de factorizar grandes números, lo que le permite construir funciones de una sola dirección con una trampilla. Otras funciones de una sola dirección se basan en la dificultad de extraer logaritmos discretos. Estos dos problemas de la teoría de números son los cimientos de los criptosistemas de clave pública más usados actualmente.

Cada participante en un criptosistema de clave pública tiene una clave $k = (k_p, k_s)$, que consiste enu na clave pública (k_p) y una clave privada o secreta (k_s). Para garantizar la seguridad del criptosistema, debería ser inviable obtener la clave privada k_s a partir de la clave pública k_p. Un algoritmo eficiente debería ser el encargado de elegir aleatoriamente ambas claves en un gran espacio de parámetros. Así, cualquiera puede usar k_p para encriptar mensajes, pero solo quien posea k_s podrá desencriptarlos.

En cuanto a las firmas digitales, supongamos que tenemos una familia de funciones con trampilla $\left(E_{k_p}\right)_{k_p \in K_P}$ donde cada función E_{k_p} es biyectiva. Sea k_p la clave pública de Alice, quien es la única de calcular la inversa $E_{k_p}^{-1}$ de E_{k_p} pues para ello se necesita la clave privada k_s de Alice. De esta forma, si Alice quiere firmar un mensaje m, tiene que calcular $E_{k_p}^{-1}(m)$, que será el valor de la firma s de m. Todo el mundo puede verificar la firma de Alice s pues cualquiera puede usar su clave pública k_p y calcular $E_{k_p}(s)$. Si $E_{k_p}(s) = m$, entonces podemos asegurarnos de que Alice realmente firmó m porque es la única que es capaz de calcular $E_{k_p}^{-1}(m)$.

Una importante aplicación de los criptosistemas de clave pública es que permiten intercambiar claves en sistemas de clave secreta. Si Alice conoce la clave pública de Bob, ella puede generar una clave de sesión, cifrarla con la clave pública de Bob y enviársela.

Algunos sistemas conocidos de clave pública son:

- *RSA*: está basado en el problema de factorización de enteros.
- *ElGamal*: está basado en el problema del logaritmo discreto.
- *McEliece*: está basado en la teoría de los códigos Goppa.
- *Curvas Elípticas*: son una generalización del sistema ElGamal y se basan en el problema de calcular logaritmos discretos en curvas elípticas.

5.4.1 *RSA*

El criptosistema RSA consiste en multiplicar dos números primos muy grandes y publicar su producto n. Una parte de la clave pública la conformará n, mientras que los factores de n se mantienen en secreto y se usarán como clave privada. La idea básica es que los factores de n no puedan recuperarse de n. Por lo que la seguridad de RSA radica en la dificultad del problema de factorización de enteros.

5.4.1.1 *Generación de claves*

Cada usuario del criptosistema RSA posee una clave pública y otra privada. Para generar este par de claves, se siguen los siguientes tres pasos:

1. Se eligen aleatoriamente dos grandes números primos distintos p y q y se calcula su producto $n = p \cdot q$.
2. Se elige un entero e tal que $1 < e < \phi(n)$ y sea primo con $\phi(n)$. La clave pública será (n, e).
3. Se calcula d que verifique $ed \equiv 1 \pmod{\phi(n)}$. La clave privada será (n, d).

Los números n, e y d se denominan *módulo*, *exponente de cifrado* y *exponente de descifrado*, respectivamente. El exponente de descifrado d se puede obtener con el algoritmo extendido de Euclides. Con este exponente es posible descifrar un texto cifrado y generar una firma digital.

5.4.1.2 *Cifrado y descifrado*

Supongamos que queremos cifrar un mensaje para Bob. Para ello, usaremos la clave pública (n, e) de Bob. Los mensajes que podemos encriptar $m \in \{0, ..., n-1\}$ se pueden considerar elementos de $\mathbb{Z}_n$.

Para cifrar un texto plano $m \in \mathbb{Z}_n$, podemos usar la función RSA

$$RSA_{n,e} : \mathbb{Z}_n \longrightarrow \mathbb{Z}_n$$
$$RSA_{n,e}(m) = m^e \pmod{n},$$

es decir, el texto cifrado c es m^e módulo n.

Para que Bob descifre el texto cifrado c, puede usar la función RSA

$$RSA_{n,d} : \mathbb{Z}_n \longrightarrow \mathbb{Z}_n,\ c \longmapsto c^d,$$

donde (n, d) es su clave secreta. De esta forma, Bob puede recuperar el texto plano, es decir, $m = c^d$ módulo n, pues las funciones de cifrado y descifrado $RSA_{n,e}$ y $RSA_{n,d}$ son inversas entre sí.

Con este procedimiento de cifrado, podemos cifrar secuencias de bit hasta $k := \lfloor \log_2 n \rfloor$ bits. Si los mensajes son más largos, podemos dividirlos en bloques de longitud k y cifrar cada uno por separado.

5.4.1.3 *Firma digital*

El criptosistema RSA también se puede usar para realizar firmas digitales. Sea (n, e) la clave pública y d el exponente de descifrado, si queremos firmar un mensaje m, le aplicamos el algoritmo de descifrado y obtenemos la *firma* de m, $\sigma = m^d$. Decimos que (m, σ) es un *mensaje firmado*. Para verificar ese mensaje, basta con calcular σ^e, donde e es la clave pública del firmante, y comprobar que coincide con m.

5.5 CRIPTOGRAFÍA POST-CUÁNTICA

Se cree que algumos sistemas criptográficos, tales como RSA, resisten los ataques de grandes ordenadores clásicos, pero no de los grandes ordenadores cuánticos. Sin embargo, se cree que algunas alternativas, como el sistema de McEliece con una clave de cuatro millones de bits, son capaces de resistir los ataques de grandes ordenadores clásicos y cuánticos.

Es por esto que surgen diversas preocupaciones, ante la amenaza de los ordenadores cuánticos se duda sobre si seguir usando RSA o simplemente cambiar a otros sistemas criptográficos que sean resistentes a dichos ordenadores. Sin embargo, esto no es tan sencillo pues necesitamos tiempo para mejorar la eficiencia, fomentar la confanza y mejorar de la usabilidad de la criptografía post-cuántica. En breve, todavía no estamos preparados para que el mundo cambie a la criptografía post-cuántica.

En esta sección estudiaremos los sistemas criptográficos basados en códigos, criptosistemas que usan un código de corrección de errores $\mathcal{C}$. Para ello, agregan un error a una palabra de $\mathcal{C}$ o al calcular un síndrome relativo a la matriz de paridad de $\mathcal{C}$.

El primero de esos sistemas es un sistema de cifrado de clave pública y fue propuesto por Robert J. McEliece in 1978. Este sistema usa como clave privada un código de Goppa binario aleatorio y, como clave pública, una matriz generadora aleatoria de una versión permutada aleatoriamente de ese código. El texto cifrado es una palabra código a la que se le han añadido algunos errores, y solo el que posee la clave privada puede eliminar esos errores. Actualmente, no se conoce ningún ataque que presente una amenaza grave a este sistema, ni los ordenadores cuánticos. Cabe destacar la seguridad y la rapidez del criptosistema de McEliece, ya que tanto los procedimientos de cifrado y descifrado son de baja complejidad.

5.5.1 *Criptosistema de McEliece*

En 1978, R. J. McEliece propuso un criptosistema de clave pública [8]. El criptosistema de McEliece se basa en códigos lineales de corrección de errores y hasta ahora es el criptosistema más exitoso basado en nociones de teoría de codificación.

La construcción original en [8] usa códigos de Goppa binarios para cifrar y descifrar mensajes. Sim embargo, han surgido otras variantes de este criptosistema que usan otros códigos lineales, pero la mayoría han resultado ser inseguros. A día de hoy, la construcción original de 1978 se considera segura con la correcta elección de parámetros.

Por esta razón, el criptosistemade McEliece se pone a la altura del RSA. Existen algunas diferencias entre ambos criptosistemas, el de McEliece es capaz de cifrar y descifrar mensajes más rápido. Sin embargo, los tamaños de las claves son mayores que los del RSA, que es la razón por la que el criptosistema de McEliece apenas se usa. La principal razón por la que está creciendo el interés en el criptosistema de McEliece es porque es uno de los mejores candidatos para criptosistemas de clave pública seguros post-cuánticos. El esquema Niederreiter, una variante del criptosistema de McEliece, también permite construir un esquema de firma digital segura.

En esta sección presentaremos el criptosistema de McEliece incluyendo algunas de sus variantes y estudiaremos su seguridad y los ataques más conocidos.

5.5.2 *Original construcción*

El criptosistema de McEliece usa códigos lineales de corrección de rrores para cifrar mensajes. Este criptosistema posee una clave privada, que se elige al generar la clave y contiene la descripción del código lineal estructurado, y una clave pública que se basa en una versión suficientemente aleatorizada de ese mismo código. De esta forma, será difícil descifrar un mensaje sin conocer la estructura del código lineal (clave privada), pues es lo que proporciona un descifrado rápido.

Los códigos que usa la construcción original son los códigos de Goppa binarios irreducibles. Estos códigos son muy adecuados pues poseen altas capacidades de corrección de erorres y matrices generadoras densas, que son difíciles de distinguir de matrices binarias aleatorias.

5.5.2.1 *Generación de claves*

La generación de la clave de este criptosistema se obtiene a partir de los siguientes pasos:

1. Se elige un $[n, k, 2t + 1]$-código lineal aleatorio $\mathcal{C}$ sobre $\mathbb{F}_2$ que tenga un algoritmo de decodificación eficiente $\mathcal{D}$ que sea capaz de corregir hasta t errores.
2. Se calcula la matriz generadora G de dimensión $k \times n$ para $\mathcal{C}$.
3. Se genera una matriz no singular binaria aleatoria S de dimensión $k \times k$.
4. Se genera una matriz de permutaciones aleatoria P de dimensión $n \times n$.
5. Se calcula la matriz $G' = SGP$ de dimensión $k \times n$. La clave pública es (G', t) y la clave privada es $(S, G, P, \mathcal{D})$.

Es decir, la matriz G' se obtiene al permutar las columnas de G a partir de la matriz P y luego realizar un cambio de base con la matriz S. De esta forma, la matriz G' corresponde a un $[n, k, 2t + 1]$-código lineal que es equivalente permutacionalmente a la clave privada elegida. Llamaremos a G' como la *matriz generadora pública*.

5.5.2.2 *Cifrado y descifrado*

Una vez tenemos la clave, podemos **cifrar** un texto plano $\mathbf{m} \in \{0,1\}^k$ eligiendo un vector aleatorio $\mathbf{e} \in \{0,1\}^n$ de peso t y calcular el texto cifrado como

$$\mathbf{c} = \mathbf{m}G' + \mathbf{e}.$$

Para recuperar $\mathbf{m}$ eficientemente, podemos usar el siguiente algoritmo de **descifrado**. Sea un criptograma $\mathbf{c} \in \{0,1\}^n$, primero calculamos

$$\mathbf{c}P^{-1} = (\mathbf{m}S)G + \mathbf{e}P^{-1}.$$

Ahora ya podemos aplicar el algoritmo de descifrado $\mathcal{D}$ a $\mathbf{c}P^{-1}$ para obtener $\mathbf{c'} = \mathbf{m}S$, pues $(\mathbf{m}S)G$ es una palabra código válida para el código lineal elegido y $\mathbf{e}P^{-1}$ tiene peso t. Finalmente, podemos calcular el mensaje $\mathbf{m}$ con

$$\mathbf{m} = \mathbf{c'}S^{-1}.$$

Por otra parte, a la hora de aplicar el algoritmo de Sugiyama, sabemos que este algoritmo de descifrado necesita saber el polinomio generador. De este modo consideraremos la clave privada como $(S, G, P, g(x))$, donde $g(x)$ es el polilnomio de Goppa para el código elegido.

5.5.3 *Criptosistema Niederreiter*

En 1986, H. Niederreiter propuso una variante relevante del criptosistema de McEliece en [9]. A diferencia del criptosistema de McEliece, este criptosistema usa una matriz de paridad en vez de una matriz generadora.

Antes de introducir este criptosistema, necesitamos definir el concepto de síndrome de una palabra en $\mathbb{F}^n$.

Definición 22. Sea $\mathcal{C}$ un $[n,k,d]$-código lineal sobre $\mathbb{F}$ y sea H su matriz de paridad. El *síndrome* de una palabra $\mathbf{y} \in \mathbb{F}^n$ se define como

$$\mathbf{s} = \mathbf{y}H^T.$$

Sabemos que una palabra código de $\mathcal{C}$ tiene síndrome igual a $\mathbf{o}$, según la definición de la matriz de paridad. Sean $\mathbf{y}_1, \mathbf{y}_2 \in \mathbb{F}^n$ dos vectores, entonces

$$\mathbf{y}_1 - \mathbf{y}_2 \in \mathcal{C} \Leftrightarrow (\mathbf{y}_1 - \mathbf{y}_2)H^T = \mathbf{o} \Leftrightarrow \mathbf{y}_1H^T = \mathbf{y}_2H^T.$$

El hecho de que $\mathbf{y}_1 - \mathbf{y}_2$ sea una palabra código de $\mathcal{C}$ si y solo si los síndromes de $\mathbf{y}_1$ e $\mathbf{y}_2$ son iguales es la base para un método eficiente para implementar la decodificación de la palabra código más cercana que se llama *decodificación del síndrome*.

Sea una palabra $\mathbf{y} \in \mathbb{F}^n$, un algoritmo de decodificación de síndrome $\mathcal{D}$ encuentra una palabra con el mínimo peso $\mathbf{e} \in \mathbb{F}^n$ tal que

$$\mathbf{y}H^T = \mathbf{e}H^T.$$

Si $\mathbf{y}$ tiene la forma $\mathbf{y} = \mathbf{c} + \mathbf{e'}$, donde $\mathbf{c} \in \mathcal{C}$ y $\mathbf{w}(\mathbf{e'}) \leq t$, entonces $\mathbf{e} = \mathbf{e'}$, esto es, el algoritmo de decodificación de síndrome encuentra exactamente el vector error que se le añadió a la palabra código.

Ahora, ya podemos definir el criptosistema Niederreiter, que se basa en la idea de la decodificación del síndrome.

5.5.3.1 *Generación de claves*

La generación de la clave de este criptosistema se obtiene a partir de los siguientes pasos:

1. Se elige un $[n, k, 2t + 1]$-código lineal aleatorio $\mathcal{C}$ sobre $\mathbb{F}_2$ que tenga un algoritmo de decodificación del síndrome eficiente $\mathcal{D}$ que sea capaz de corregir hasta t errores.
2. Se calcula la matriz de paridad H de dimensión $(n - k) \times n$ para $\mathcal{C}$.
3. Se genera una matriz no singular binaria aleatoria S de dimensión $(n - k) \times (n - k)$.
4. Se genera una matriz de permutaciones aleatoria P de dimensión $n \times n$.
5. Se calcula la matriz $H' = SHP$ de dimensión $(n - k) \times n$. La clave pública es (H', t) y la clave privada es $(S, H, P, \mathcal{D})$.

5.5.3.2 *Cifrado y descifrado*

Una vez tenemos la clave, podemos **cifrar** un texto plano $\mathbf{m} \in \{0, 1\}^k$ con peso t, calculando el texto cifrado como el síndrome de $\mathbf{m}$

$$\mathbf{c} = \mathbf{m}H'^T.$$

Para recuperar $\mathbf{m}$ eficientemente, podemos usar el siguiente algoritmo de **descifrado**. Sea un criptograma $\mathbf{c}$, primero calculamos

$$S^{-1}\mathbf{c}^T = HP\mathbf{m}^T.$$

Luego, buscamos un vector $\mathbf{z} \in \mathbb{F}^n$ tal que $H\mathbf{z}^T = HP\mathbf{m}^T$. Entonces $\mathbf{z} - (P\mathbf{m}^T)^T = \mathbf{z} - \mathbf{m}P^T$ es una palabra código válida en $\mathcal{C}$ debido a que los síndromes de las palabras son iguales. Como $\mathbf{m}P^T$ tiene peso t, podemos aplicar $\mathcal{D}$ a $\mathbf{z}$ para encontrar el vector error $\mathbf{m}P^T$ y por lo tanto $\mathbf{m}$.

Observemos que el mensaje en texto plano se representa como el error de la palabra código en lugar de la palabra con la información original. De este modo, el mensaje en texto plano debe codificarse adicionalmente en un vector de peso t para el cifrado.

A diferencia con el criptosistema de McEliece, el algoritmo de decodificación del síndrome es más eficiente que el algoritmo de decodificación del criptosistema de McEliece. Además, tambien se puede usar para construir un esquema de firma digital.

Sin embargo, si un ataque es capaz de romper el criptosistema de McEliece, también puede romper el esquema Niederreiter y viceversa. Esto se debe a que los criptosistemas de McEliece y Niederreiter se basan en el uso de códigos lineales que son duales entre si y la matriz generadora se puede obtener eficientemente a partir de la matriz de paridad y viceversa.

5.5.4 *Seguridad del criptosistema de McEliece*

En esta sección estudiaremos la seguridad del criptosistema de McEliece original y analizaremos los ataques más conocidos ante este criptosistema. Como la seguridad de los esquemas de McEliece y Niederreiter es equivalente [10], luego todos los ataques que analicemos también estarán relacionados implícitamente con el esquema Niederreiter.

Fijemos un código de Goppa $\Gamma(L, g) \subset \mathbb{F}_2^n$, con $g(x) \in \mathbb{F}_{2^m}[x]$ y L una tupla de n elementos distintos de $\mathbb{F}_{2^m}$, capaz de corregir hasta t errores. Tenemos que la dimensión del código es $k = n - tm$. Sea G la matriz generadora binaria de dimensión $k \times n$ de Γ y definimos $G' = SGP$ como la clave pública de McEliece, donde S es la matriz binaria no singular de dimensión $k \times k$ y P es la matriz de permutaciones de dimensión $n \times n$.

Para estudiar la seguridad de este criptosistema, tenemos que determinar cómo de difícil es determinar el mensaje $\mathbf{m}$ a partir de conocer G' y haber interceptado $\mathbf{c}$. Como se indica en [8], pueden darse dos ataques básicos:

1. Intentar recuperar la clave secreta G a partir de G' y así descifrar el mensaje.
2. Intentar recuperar el mensaje original $\mathbf{m}$ a partir del criptograma $\mathbf{c}$ sin conocer la clave privada G.

El primer ataque parece inviable si n y t son suficientemente grandes, pues existen demasiadas posibilidades tanto para G como para S y P.

El segundo ataque puede ser más prometedor para el adversario pues puede aproximarse mediante la decodificación de conjuntos de información.

La seguridad del criptosistema de McEliece se puede insinuar por la intratabilidad de los siguientes problemas fundamentales en la teoría de la codificación.

Problema 3 (Problema general de decodificación para códigos lineales)**.** *Sea $\mathcal{C}$ un $[n, k]$-código lineal sobre $\mathbb{F}$ e $\boldsymbol{y} \in \mathbb{F}^n$. Encontrar una palabra código $\boldsymbol{c} \in \mathcal{C}$ tal que la distancia $d(\boldsymbol{y}, \boldsymbol{c})$ sea mínima.*

Problema 4 (Problema de encontrar una palabra código dado un peso)**.** *Sea $\mathcal{C}$ un $[n, k]$-código lineal sobre $\mathbb{F}$ y $w \in \mathbb{N}$. Encontrar una palabra código $\boldsymbol{c} \in \mathcal{C}$ tal que $\boldsymbol{w}(\boldsymbol{c}) = w$.*

Se ha demostrado que ambos problemas son NP-duro [11]. No obstante, esto no implica que romper el criptosistema de McEliece sea NP-duro, pues los códigos binarios de Goppa solo cubren una fracción de todos los códigos lineales posibles. Es por esto que la seguridad del

criptosistema de McEliece se basa en la suposición de que la clave pública es indistinguible de cualquier matriz aleatoria.

5.5.4.1 *Seguridad post-cuántica*

El criptosistema de McEliece es inmune ante el algoritmo de Shor [12], que si se implementara en un ordenador cuántico podría romper otros criptosistemas de clave pública tales como RSA.

6

IMPLEMENTACIÓN EN SAGEMATH DEL/DE LOS CRIPTOSISMEAS DE MCELIECE/NIEDERREITER

7

ATAQUE USANDO ALGORITMOS GENÉTICOS

A

IMPLEMENTACIÓN EN SAGEMATH DE LOS CÓDIGOS GOPPA

B

IMPLEMENTACIÓN EN SAGEMATH DEL CRIPTOSISTEMA DE MCELIECE

C

IMPLEMENTACIÓN EN SAGEMATH DEL CRIPTOSISTEMA DE NIEDERREITER

CONCLUSIÓN

Añadir conclusión

BIBLIOGRAFÍA

[1] W. Cary Huffman and Vera Pless. *Fundamentals of Error-Correcting Codes*. Cambridge University Press, 2010. ISBN 978-0-521-13170-4. URL http://www.cambridge.org/9780521782807.

[2] Podestá and Ricardo. Introducción a la teoría de códigos autocorrectores. page 50, 2006. URL https://www.famaf.unc.edu.ar/documents/940/CMat35-3.pdf.

[3] Alexander Vardy. The intractability of computing the minimun distance of a code. *IEEE Transactions on Information Theory*, 43(6), 1997.

[4] Anton Betten, Michael Braun, Harald Fripertinger, Adalbert Kerber, Axel Kohnert, and Alfred Wassermann. *Error-Correcting Linear Codes*. Springer, 2006. ISBN 978-3-540-28371-5. URL http://www.cambridge.org/9780521782807.

[5] Elwyn R. Berlekamp. Goppa codes. *IEEE Transactions on Information Theory*, 19(5), 1973.

[6] M. A. Tsfasman, S. G. Vladut, and T. Zink. Modular curves, shimura curves, and goppa codes, better than varshamov-gilbert bound. *Mathematische Nachrichten*, 109(1):21–22, 1982.

[7] Y. Sugiyama, M. Kasahara, Y. Hirashawa, and T. Namekawa. A method for solving key equation for decoding goppa codes. *Information and Control*, 27(1):87–99, 1975.

[8] R. J. McEliece. A public-key cryptosystem based on algebraic coding theory. pages 114–116, 1978. URL https://tmo.jpl.nasa.gov/progress_report2/42-44/44N.PDF.

[9] H. Niederreiter. Knapsack-type cryptosystems and algebraic coding theory. *Problems of Control and Information Theory*, 15(2):159–166, 1986.

[10] Y. Li, R. H. Deng, and X. Wang. On the equivalence of mceliece's and niederreiter's public-key cryptosystems. *IEEE Transactions on Information Theory*, 40(1), 1994.

[11] E. Berlekamp, R. McEliece, and H. Van Tilborg. On the inherent intractability of certain coding problems. *IEEE Transactions on Information Theory*, 24(3):384–386, 1978.

[12] P. W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM J. Comput.*, 26(5):1484–1509, 1997.