Introducción a la Criptografía post-cuántica basada en teoría de códigos

Autor

Daniel Felipe Rambaut Lemus

Trabajo presentado como requisito para optar por el título de Profesional en Matemáticas aplicadas y ciencias de la computación

Directora

Valérie Gauthier Umaña

Escuelas de Ingeniería Ciencia y Tecnología

Matemáticas aplicadas y ciencias de la computación

Universidad del Rosario

Bogota-Colombia

2021

Resumen

La criptografía es la disciplina que estudia el arte de transformar un mensaje (llamado texto plano) en un mensaje no legible por un tercero (llamado texto cifrado) utilizando una clave secreta. Los protocolos de cifrado, descifrado y construcción de claves es lo que llamamos un criptosistema. Existen dos grandes familias:

1. Criptografía simétrica: conformada por los criptosistemas que utilizan una misma clave secreta para cifrar y descifrar mensajes.

2. Criptografía asimétrica o a clave pública: son aquellos en los que los procesos de cifrado y descifrado son llevados a cabo por dos claves, una pública para el proceso de cifrado y otra secreta para descifrado.

La criptografía simétrica tiene dos principales problemas: las personas que van a comunicarse deben tener un primer encuentro para definir la clave secreta y por otro lado para cada pareja de personas que se quieran comunicar debe existir una clave secreta. Ambos problemas son resueltos por la criptografía a clave pública ya que en este caso no hay necesidad de ponerse de acuerdo con la clave y por otro lado una misma clave pública le permite a una persona recibir mensajes de muchas personas sin que estas tengan la posibilidad de descifrar el mensaje. Esto hace que la criptografía a clave pública sea la base del comercio electrónico.

En 1978 se propuso el RSA [36], el cual fue el primer criptosistema a clave pública. Más de 40 años después los criptosistema a clave pública que se utilizan dependen únicamente de dos problemas matemáticos: la factorización y el logaritmo discreto.

Es decir que si de alguna manera lográramos resolver estos problemas, toda la criptografía a clave pública quedaría expuesta e insegura. Sin embargo en 1994, Pete Shor, publicó un algoritmo [40] en el cual, de tener un computador cuántico suficientemente poderoso, podría resolver estos dos problemas. La carrera de varias empresas y centros de investigación por crear un computador cuántico ha sido bastante activa y ya se han creados algunos en los cuales se ha podido implementar el algoritmo de Shor y factorizar, por ejemplo, el número 15 [21].

Como respuesta a este nuevo escenario, en donde la computación cuántica pone en jaque a toda la criptografía a clave pública, se presenta la criptografía post-cuántica, la cual consiste en buscar criptosistemas que sean resistentes a ataques hechos en computadores convencionales y cuánticos.

El instituto Nacional de estándares y Tecnología de Estados Unidos (llamado NIST por sus siglas en inglés, National Institute of Standards and Technology) preocupado por esta situación, y buscando promover la investigación en critpografía post-cuántica organizó un concurso público para buscar un criptosistma post-cuántico que se pueda convertir en el estándar. Existen diferentes familias que han inspirado algunas propuestas de criptosistemas post-cuánticos: la teoría de códigos, retículos funciones de Hash y álgebra multivariada que se vienen estudiando aproximadamente desde los años 2000 y recientemente se trabajan con *isogeny* en curvas elípticas.

En este trabajo de grado nos concentramos en la criptografía post-cuántica basada en la teoría de códigos. En 1978, McEliece propuso un criptosistema [30] que no tuvo mucha acogida dado su tamaño de la clave secreta, pero que resulta ser resistente a ataques post-cuánticos. En los últimos 20 años se han propuesto varias variantes del McEliece, que usan la misma idea de basarse en códigos correctores de errores, pero que usan protocolos diferentes para tratar de reducir el tamaño de la clave. Hasta el momento la mayoría han sido atacados, existen algunos vigentes, pero todavía la comunidad no tiene confianza en su seguridad ya que son muy

recientes.

En esta tesis se realizó un documento donde se introduce las bases matemáticas, la criptografía, la teoría de corrección de errores y la computación cuántica necesaria para poder entender la criptografía post-cuántica basada en teoría de códigos. Al final de la tesis introducimos los criptosistemas de McEliece y Niederreiter así como la versión del criptosistema de McEliece que llegó a la última etapa de la competencia de la NIST (todavía en curso) [4].

Agradecimientos

Estoy muy agradecido con la profesora Valérie Gauthier por supervisarme, por su apoyo continuo y compartir su conocimiento. Fue un gran ambiente de trabajo y de aprendizaje, estoy particularmente agradecido por haber tenido la oportunidad de trabajar y estudiar de tal manera que pudiera alcanzar mis objetivos. Gracias, Valérie por la confianza que tienes en mí.

Me gustaría agradecer a mis compañeros y amigos con los que he compartido este último año, en particular a la profesora Margot Salas y más. Con todos ellos viví un ambiente de trabajo muy amistoso y motivador, guardo muy buenos recuerdos de nuestros trabajos, ya sea realizando ejercicios de clase o realizando investigaciones. Me gustaría agradecer al profesor Julián Rincón por su colaboración en el capítulo de computación cuántica, ya que sin su ayuda este capítulo no tendría ese toque científico e informativo como se planteó desde un principio.

Agradezco a la profesora Diana Bueno por aceptar ser el jurado de mi tesis y además por sus correcciones, las cuales considero que enriquecen el contenido y logra hacer más claro lo que se quiere transmitir en esta.

Por último, pero no menos importante, me gustaría agradecer a mi familia, mi madre, padre, hermanos, mis tías, mi abuela y a Katherin Reyes quienes me apoyaron durante todo el camino y me brindaron mucha motivación y amor para lograr conseguir la meta que tanto deseaba desde que empecé la carrera.

Índice general

Índice de figuras

Índice de cuadros

1 Bases Matemáticas

En esta tesis, necesitamos hacer uso de algunas estructuras algebraicas que definiremos en esta sección. Para la mayoría de las definiciones y demostraciones nos inspiramos de los libros [5] y [23], los cuales son muy interesantes para los temas que trataremos en esta tesis. En el Apéndice A1 se presentan algunas notaciones de términos matemáticos que usaremos en este documento.

1.1. Introducción a los campos finitos

Definición 1.1.1. *[23] Un campo* $\mathbb{F}$ *es un conjunto de elementos* S *con dos operaciones binarias que denotaremos "*$+$*" y "*$\times$*" y dos elementos diferentes de* S *que llamaremos* 0 *y* 1*, tal que se satisfacen los siguientes axiomas para todo* $a, b, c \in S$*.*

1. *Clausura para las operaciones "*$+$*" y "*$\times$*"*
2. *Operación "*$+$*":*
   a) *Asociatividad:* $a + (b + c) = (a + b) + c$*.*
   b) *Conmutatividad:* $a + b = b + a$*.*
   c) *Identidad:* $a + 0 = 0 + a = a$*.*
   d) *Inverso: existe un elemento* $d \in S$ *tal que* $a + d = d + a = 0$*.*
3. *Operación "*$\times$*":*
   a) *Asociatividad:* $a \times (b \times c) = (a \times b) \times c$*.*

b) *Conmutatividad:* $a \times b = b \times a$.

c) *Identidad:* $a \times 1 = 1 \times a = a$.

d) *Inverso: para todo* $a \neq 0$ *en* S *existe un elemento* $d \in S$ *tal que* $a \times d = d \times a = 1$.

4. *Distributiva:* $a \times (b + c) = a \times b + a \times c$.

Algunos ejemplos de campos infinitos son $\mathbb{Z}, \mathbb{R}$ y $\mathbb{C}$. Si el campo contiene solo un número finito de elementos, es llamado un *campo finito*. Consideremos, por ejemplo, los enteros módulo 3, denotados por $\mathbb{Z}_3$, cuyos elementos son $\{0, 1, 2\}$ y las operaciones suma y multiplicación definidas de la siguiente manera.

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

$\times$	0	1	2
0	0	0	0
1	0	1	2
2	0	2	1

Hay que tener cuidado ya que no se puede generalizar diciendo que para todo $n \in \mathbb{N}^*$, $\mathbb{Z}_n$ es un campo. De hecho, si tomamos $n = 9$, podemos ver que el elemento 3 en $\mathbb{Z}_9$ no tiene inverso multiplicativo. El siguiente resultado de teoría de números es muy interesante y nos da muchos ejemplos de campos finitos.

Teorema 1.1.1. *[23]* $\mathbb{Z}_p$ *es un campo finito si y solo si* p *es un número primo.*

Demostración. Para demostrar la primera dirección, vamos a utilizar la contraposición. Demostremos entonces que si p no es primo, $\mathbb{Z}_p$ no es un campo. Asumamos entonces que p no es primo, existen entonces $a, b \in \mathbb{N}^*$ tal que $p = a \times b$. Sabemos que $a, b < p$, por lo tanto pertenecen a $\mathbb{Z}_p$. Supongamos que b tiene un inverso multiplicativo c, tenemos entonces que $b \times c \equiv 1 \mod p$ esto implica que $a \times b \times c \equiv a \mod p$. Por lo tanto $0 \equiv a \mod p$, lo cual es falso. Podemos concluir que b no es invertible y entonces $\mathbb{Z}_p$ no es un campo.

Ahora demostremos la segunda dirección: si p es primo, entonces la mayoría de los axiomas de campo son fácil de verificar y son dejados al lector. Sin embargo, el axioma de la existencia del inverso multiplicativo no es tan trivial. Sea a un elemento tal que $0 < a < p$, sabemos que el $gcd(p, a) = 1$, ya que p es un número primo. Por el algoritmo extendido de Euclides encontramos dos enteros s y t tales que

$$s \times p + t \times a = 1.$$

Tomando el modulo p de la ecuación, encontramos que $t \times a \equiv 1 \mod p$ es decir que t es el inverso de a modulo p terminando así nuestra demostración. □

De ahora en adelante, usaremos la siguiente notación: $\mathbb{F}_q$ para denotar un campo finito con q elementos y 1 para denotar su identidad. Sea $a \in \mathbb{F}_q$ diferente de cero, denotaremos a^i al resultado de la operación $a \times a, \ldots, \times a$, i veces. Por convención, denotaremos $a^0 = 1$.

Definición 1.1.2. *[23] Sea $a \in \mathbb{F}_q \setminus \{0\}$. Definimos el orden de a, el cual denotamos $ord(a)$, como el menor entero positivo s, distinto de 0, tal que $a^s = 1$.*

Verifiquemos que la definición de orden está bien definida y por lo tanto el orden de a existe. Para esto, consideremos el conjunto de todas las potencias de a:

$$A = \{a^i : i = 1, 2, \ldots\}.$$

Recordemos que $\mathbb{F}_q$ tiene q elementos y es cerrado bajo multiplicación, por lo tanto, A es un subconjunto de $\mathbb{F}_q$ y a lo sumo tiene q elementos. Esto quiere decir que existen al menos dos enteros, distintos de cero, i y j, diferentes entre ellos, tales que $a^i = a^j$ y por lo tanto $a^{i-j} = 1$. Es decir, existe al menos un elemento t, distinto de cero tal que $a^t = 1$ y entonces el orden de a existe.

Lema 1.1.1. *[23] Sea $\mathbb{F}_q$ un campo finito y sean a y b dos elementos de $\mathbb{F}_q \setminus \{0\}$. Las siguientes afirmaciones son correctas.*

1. *Si* $ord(a) = s$ *entonces* $a, a^2, \ldots, a^s$ *son elementos distintos entre ellos.*
2. $a^j = 1 \Leftrightarrow ord(a)|j$
3. $ord(a^j) = \frac{ord(a)}{gcd(ord(a),j)}$
4. *Si* $ord(a) = s$, $ord(b) = j$ *y* $gcd(s, j) = 1$ *entonces* $ord(ab) = sj$

Demostración.

1. Demostraremos esta propiedad por contradicción. Supongamos que existen i, j tales que $a^i = a^j$ para $0 < i < j \leq s$. Si multiplicamos a la derecha por el inverso de a^i, obtenemos que

$$1 = a^i \times (a^i)^{-1} = a^j \times (a^i)^{-1}$$

   por hipótesis $0 < j - i < s$, lo cual contradice que el orden de a sea s. En consecuencia los elementos $a, a^2, \ldots, a^s$ son todos diferentes.

2. Supongamos que el $a^j = 1$ y $ord(a) = s$. Entonces si tomamos $j = sh + r$ con $0 \leq r < s$, tenemos

$$1 = a^j = a^{sh+r} = (a^s)^h \times a^r = 1^h \times a^r = a^r.$$

   Por tanto, por definición de orden se tiene $r = 0$. En consecuencia $s|j$.
   Por otro lado si suponemos que $j = sh$, tenemos

$$a^j = a^{sh} = (a^s)^h = 1.$$

3. Supongamos $ord(a) = s$ y $ord(a^j) = l$. Entonces

$$1 = (a^j)^l = a^{jl}.$$

   Aplicando el resultado anterior, $s|jl$ y por lo tanto $\frac{s}{gcd(s,j)} | \frac{jl}{gcd(s,j)}$. En conse-

cuencia, se tiene que $\frac{s}{gcd(s,j)}|l$.

Por otro lado, tenemos que

$$(a^j)^{\frac{s}{gcd(s,j)}} = (a^s)^{\frac{j}{gcd(s,j)}} = 1.$$

Aplicando el resultado anterior, se tiene que $l|\frac{s}{gcd(s,j)}$. En consecuencia

$$ord(a^j) = \frac{s}{gcd(s,j)}.$$

4. Supongamos $ord(a) = s$, $ord(b) = j$ y $gcd(s,j) = 1$, entonces tomando

$$(ab)^{sj} = (a)^{sj}(b)^{sj} = (a^s)^j(b^j)^s = 1 \times 1 = 1.$$

Aplicando 2, tenemos $ord(ab)|sj$, pero como el $gcd(s,j) = 1$ implica que

$$ord(ab) = l_1 \times l_2$$

donde $l_1|s$, $l_2|j$. Para terminar la demostración debemos demostrar que $l_1 = s$ y $l_2 = j$. Para ello consideremos

$$1 = (ab)^{l_2 l_1} = [(ab)^{l_1 l_2}]^{\frac{s}{l_1}} = [a^{l_1 l_2} b^{l_1 l_2}]^{\frac{s}{l_1}} = a^{s l_2} b^{s l_2} = 1 \times b^{s l_2} = b^{s l_2}.$$

Por 2, tenemos $j|l_2 s$ y ya que $gcd(s,j) = 1$ se tiene que $j|l_2$, por tanto $j = l_2$. Análogamente se puede obtener que $s = l_1$. En consecuencia $ord(ab) = sj$.

□

El Lema 1.1.1 nos permite demostrar el siguiente teorema.

Teorema 1.1.2. *$\mathbb{F}_q$ tiene un elemento de orden $q - 1$.*

Demostración. Consideremos el conjunto $\mathbb{F}_q \setminus \{0\}$, el cual tiene cardinalidad $q - 1$, entonces por el punto 1 del Lema 1.1.1 se tiene que el orden de algún elemento es a

lo sumo $q-1$.

Definamos $\alpha, \beta \in \mathbb{F}_q \setminus \{0\}$ tales que α es un elemento con orden máximo. Consideremos $ord(\alpha) = r$ y $ord(\beta) = s$, queremos demostrar que $s|r$. Por contradicción se tiene que existe un numero p primo e $i, j \in \mathbb{N}$ tales que $r = p^i \times a$, $s = p^j \times b$, con $j > i$ y $gcd(a, p) = gcd(b, p) = 1$. Aplicando el punto 3 del Lema 1.1.1, tenemos

$$ord(\alpha^{p^i}) = \frac{r}{gcd(r, p^i)} = a \text{ y } ord(\beta^b) = \frac{s}{gcd(s, b)} = p^j$$

Ya que $gcd(a, p) = 1$ podemos aplicar el punto 4 del Lema 1.1.1, lo cual nos da

$$ord(\alpha^{p^i} \times \beta^b) = a \times p^j$$

Sin embargo, observe que $a \times p^j > a \times p^i = r$ lo cual contradice la hipótesis sobre r. Así tenemos que $s|r$. Esto implica que cada elemento en $\mathbb{F} \setminus \{0\}$ es un cero de el polinomio $z^{ord(\alpha)} - 1$ y ya que un polinomio de grado n puede tener a lo máximo n ceros, concluimos que el $ord(\alpha) \geq q-1$ y por tanto $ord(\alpha) = q-1$ y en consecuencia el teorema es demostrado. □

Colorario 1.1.1. *[23] Sea $\alpha \neq 0$ un elemento de $\mathbb{F}_q$ con $ord(\alpha) = t$, entonces $t|q-1$.*

Demostración. El punto 2 del Lema 1.1.1, nos dice que $t|q-1$ es equivalente a $\alpha^{q-1} = 1$.

Por el Teorema 1.1.2 sabemos que existe $\beta \in \mathbb{F}_q$ de orden $q-1$. Por el punto 1 del Lema 1.1.1, tenemos $\beta, \beta^2, \ldots, \beta^{q-1}$ elementos distintos. Tomando $\alpha = \beta^i$ para algún i, se tiene

$$\alpha^{q-1} = (\beta^i)^{q-1} = (\beta^{q-1})^i = 1^i = 1$$

.

Luego t divide a $q-1$. □

Colorario 1.1.2. *[23] Algún elemento α de un campo finito con q elementos satisface la siguiente ecuación $\alpha^q - \alpha = 0$.*

Demostración. Supongamos $\alpha \in \mathbb{F}$. Observe que ya se demostró que

$$\alpha^{q-1} = 1.$$

Entonces multiplicando por α, se tiene que $\alpha^q = \alpha$. Por lo tanto, $\alpha^q - \alpha = 0$. □

Teorema 1.1.3. *Sea $c \in \mathbb{F}_{2^m}$ entonces existe un a tal que $a \cdot a = c$.*

Demostración. Sabemos que $\mathbb{F}_{2^m}$ es un campo y por tanto existe d tal que $d \cdot c = 1$. Además como $\mathbb{F}_{2^m}$ es un campo finito entonces existe un t tal que sea el orden de d, es decir $d^t = 1$ y sabemos por el Colorario 1.1.1 que $t|(2^m - 1)$. Observe que $2^m - 1 = t \cdot k$ con $k \in \mathbb{Z}$. De lo anterior podemos ver que t es impar, por lo que existe un $s \in \mathbb{Z}$ tal que $t = 2 \cdot s + 1$. Luego

$$\begin{aligned} d^t &= d^{2\cdot s+1} = 1 \\ &= d^{2\cdot s} \cdot d \cdot c = 1 \cdot c \\ &= d^s \times d^s = c \end{aligned}$$

Por tanto, podemos ver que $a = d^s$. □

Teorema 1.1.4. *La transformación $\mathbb{F}_{2^m} \to \mathbb{F}_{2^m}$, definida como $x \to x^2$ es el automorfismo de Frobenius en $\mathbb{F}_{2^m}$.*

Demostración. La demostración en detalle se encuentra en [13]. □

Cuando se trabaja en un campo finito necesitamos una forma fácil de poder sumar y multiplicar. Una forma simple de sumar elementos sobre un campo es escribiéndolas como m tuplas y sumando usando la adición vectorial. Desafortunadamente, la multiplicación de m tuplas no es tan sencillo. Necesitamos una forma sencilla

de realizar la multiplicación, es decir, necesitamos una forma de conectar los elementos del campo con un conjunto de la forma $m-$tuplas. El siguiente teorema nos ayuda con este problema. Recordemos que el conjunto $\mathbb{F}_q^*$ es el grupo con elementos diferentes de cero.

Teorema 1.1.5. *[23]*

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

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

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

Cada generador γ de $\mathbb{F}_q^*$ es llamado elemento primitivo de $\mathbb{F}_q$. Cuando los elementos son diferentes de cero del campo finito, los podemos expresar como potencias de γ, la multiplicación en el campo es fácil de llevar acabo, ya que acorde a la regla de la multiplicación $\gamma^i\gamma^j = \gamma^{i+j} = \gamma^s$, donde $0 \geq s \geq q-2$ y $i + j \equiv s \mod q-1$.

1.2. Introducción a polinomios

En esta sección definiremos los polinomios y dos de sus operaciones, suma y multiplicación. Consideremos $\mathbb{F}$ un campo.

Definición 1.2.1. *[5] Un polinomio con coeficientes en $\mathbb{F}$ es una expresión de la forma*

$$f(x) = a_0 + a_1x + \ldots + a_nx^n + \ldots$$

Donde los coeficientes a_i son elementos de $\mathbb{F}$ para todo $i \in \mathbb{N}$ y solo un número finito de coeficientes son diferentes de cero.

Definición 1.2.2. *[5] El grado de un polinomio $f(x)$, denotado $deg(f(x))$, es la potencia más alta de x tal que a_n sea distinto de cero.*

Definición 1.2.3. *Sea $f(x)$ un polinomio de grado n, el monomio $a_n x^n$ es llamado el término principal de $f(x)$.*

Propiedad 1.2.1.

Cada polinomio es único y es definido por sus coeficientes. Si definimos

$$g = b_0 + b_1 + \ldots + b_m x^m + \ldots$$

entonces decimos que $f = g$ si y solo si $n = m$ y $a_i = b_i$ para todo $i \in [0, n]$.

El conjunto de todos los polinomios con coeficientes en $\mathbb{F}$ lo denotamos $\mathbb{F}[x]$. Podemos sumar y multiplicar polinomios en $\mathbb{F}[x]$ de la siguiente manera.

Definición 1.2.4. *[5] Sean f y g dos polinomios de $\mathbb{F}[x]$ de grado n y m respectivamente. Asumamos, sin perder generalidad, que $m < n$, y definimos la suma de los dos polinomios de la siguiente manera:*

$$f(x) + g(x) = (a_0 + b_0) + (a_1 + b_1)x + \ldots + (a_n + b_n)x^n.$$

Sean ax^m y bx^n dos monomios distintos, su multiplicación está definida por

$$ax^m \times bx^n = abx^{m+n}.$$

Usando la ley de distribución, la multiplicación de los polinomios $f(x)$ y $g(x)$ está dada por la siguiente definición.

Definición 1.2.5. *[5] Sean f y g dos polinomios de $\mathbb{F}[x]$. Definimos la multiplicación de la siguiente manera:*

$$f \times g = \left(\sum a_i x^i\right) \times \left(\sum b_i x^i\right) = \sum d_i x^i.$$

Donde $d_i = \sum_{j=0}^{i} a_j b_{i-j}$. Para poder definir la división entre polinomios, veamos el siguiente resultado.

Teorema 1.2.1. *[23] Sean $f(x), g(x) \in \mathbb{F}[x]$ con $f(x) \neq 0$. Existen dos polinomios únicos $q(x)$ y $r(x)$, tales que $deg(r(x)) < deg(f(x))$ y $g(x) = f(x)q(x) + r(x)$.*

Demostración. Primero demostraremos la existencia de los polinomios $q(x)$ y $r(x)$. Para ello, consideremos los siguientes casos.

1. Si $\deg g(x) < \deg f(x)$, tomando $q(x) = 0$ y $r(x) = g(x)$, tenemos que el polinomio $g(x) = f(x)q(x) + r(x)$ y $\deg r(x) < \deg f(x)$.

2. Si $deg(g(x)) \geq deg(f(x))$, definamos n el grado de $g(x)$ y d el grado de $f(x)$, tenemos entonces que $n \geq d$. Sea $f(x) = a_d x^d + \ldots + a_0$ de grado d con $a_d \neq 0$ y $g(x) = b_n x^n + \ldots + b_0$ con $b_n \neq 0$.
   Usemos inducción sobre el grado de $g(x)$ para demostrar la existencia de los polinomios $q(x)$ y $r(x)$.

   Caso base: Supongamos que $n = 1$, luego $d = 1$. Por tanto,

   $$f(x) = a_1 x + a_0 \text{ y } g(x) = b_1 x + b_0.$$

   Observe que si tomamos $q(x) = (b_1 a_1^{-1})$ y $r(x) = -b_1 a_1^{-1} a_0 + b_0$, entonces

   $$\begin{aligned} f(x)q(x) + r(x) &= (a_1 x + a_0)(b_1 a_1^{-1}) - b_1 a_1^{-1} a_0 + b_0 \\ &= a_1 x b_1 a_1^{-1} + b_1 a_1^{-1} a_0 - b_1 a_1^{-1} a_0 + b_0 \\ &= b_1 x + b_0. \end{aligned}$$

   Adicionalmente $\deg(r(x)) < \deg(f(x))$, por lo tanto, el caso base es cierto.

   Hipótesis inductiva: Supongamos que la propiedad es cierta para todo polinomio $g(x)$ de grado menor o igual a n.

Como $n \geq 0$ entonces existe un $s \geq 0$ tal que $n = d + s$. Adicionalmente sabemos que a_d es diferente de cero y como $\mathbb{F}$ es un campo, existe su inverso a_d^{-1}.

Definamos $q_0(x) = cx^s$, donde $c = (b_n a_d^{-1})$, y

$$g_1(x) = g(x) - q_0(x) \times f(x). \tag{1}$$

Tenemos entonces $\deg g_1(x) < \deg g(x)$. Por lo tanto, por hipótesis de inducción, existen los polinomios $q_1(x)$ y $r(x)$ tal que $deg(r(x)) < deg(f(x))$ y

$$g_1(x) = f(x)q_1(x) + r(x).$$

Ahora, despejando $g(x)$ de (1), tenemos que

$$\begin{aligned} g(x) &= g_1(x) + q_0(x)f(x) \\ &= q_1(x)f(x) + r(x) + q_0(x)f(x) \\ &= (q_1(x) + q_0(x))f(x) + r(x) \\ &= q(x)f(x) + r(x). \end{aligned}$$

Donde $q(x) = q_1(x) + q_0(x)$. Demostrando así la existencia de los polinomios $q(x)$ y $r(x)$ tal que $\deg r(x) < \deg f(x)$ y $g(x) = f(x)q(x) + r(x)$.

Ahora vamos a demostrar que los polinomios $q(x)$ y $r(x)$ son únicos. Supongamos que existen los polinomios $q(x)$ y $q_1(x)$ distintos entre ellos y los polinomios $r(x)$ y $r_1(x)$ distintos entre ellos tales que

$$\begin{cases} \deg r(x) < \deg f(x) \\ \deg r_1(x) < \deg f(x) \\ g(x) = f(x)q(x) + r(x) \\ g(x) = f(x)q_1(x) + r_1(x). \end{cases} \tag{2}$$

Observe entonces que

$$f(x)(q(x) - q_1(x)) = r_1(x) - r(x).$$

Sea $t(x) = q(x) - q_1(x)$, como $q(x) \neq q_1(x)$, entonces el grado de $t(x)$ es distinto de 0 y por lo tanto $\deg f(x)t(x) \geq \deg f(x)$, sin embargo se tiene que $\deg(r_1(x) - r(x)) < \deg f(x)$. Lo cual contradice la ecuación (2) y por lo tanto $q(x) = q_1(x)$ y $r_1(x) = r(x)$.

□

Definición 1.2.6. *[5] Sea $f(x) \in \mathbb{F}[x]$, llamamos reducible a $f(x)$ si y solo si existen los polinomios $g(x), h(x) \in \mathbb{F}[x]$ de grado al menos uno tal que $f(x)$ se puede expresar como multiplicación de $g(x)$ y $h(x)$. Por otro lado llamamos irreducible a $f(x)$ si $deg(g(x)) = 0$ o $deg(h(x)) = 0$.*

De la definición 1.2.6, podemos construir un campo con un número finito de elementos de la siguiente manera. Dado un polinomio irreducible $f(x) \in \mathbb{F}[x]$, definimos dos polinomios $h(x), g(x) \in \mathbb{F}[x]$ que son congruentes modulo $f(x)$ si y solo si $f(x)$ divide $h(x) - g(x)$, es decir, $h(x)$ y $g(x)$ tienen el mismo residuo cuando se dividen por $f(x)$. Entonces la relación de congruencia divide a $\mathbb{F}[x]$ en clases de equivalencia, las cuales contienen a $g(x)$ y las denotamos por $[g(x)]$ y definidas como

$$[g(x)] = [h(x) : h(x) = g(x) \mod f(x)].$$

Sea $R = \mathbb{F}[x]/ < f(x) >$ el conjunto de clases de equivalencia, es decir.

$$\mathbb{F}[x]/ < f(x) >= \{[g(x)] : g(x) \in \mathbb{F}[x]\}.$$

Definimos la suma y la multiplicación de las clases de equivalencia con las siguientes reglas.

$$[g(x)] + [h(x)] = [g(x) + h(x)]$$
$$[g(x)] \times [h(x)] = [g(x) \times h(x)].$$

Entonces $\mathbb{F}[x]/ < f(x) >$ es un campo llamado el campo de polinomios sobre $\mathbb{F}$ modulo $f(x)$. Cualquier polinomio en la clase $[g(x)]$ se puede usar para representar la clase y usualmente se toma el polinomio de menor grado como el representante de dicha clase. Entonces las clases de equivalencia representan todos los polinomios en $\mathbb{F}[x]$ de grado menor que $deg(f(x))$, correspondiente a todos los posibles residuos después de dividir por $f(x)$.

1.3. Introducción a los campos finitos de orden $q = p^m$

Para comenzar vamos a suponer un polinomio $f(x) \in \mathbb{F}_p[x]$ el cual es irreducible sobre $\mathbb{F}_p$. Supongamos que $f(x)$ tiene grado m. Como vimos en la sección anterior el conjunto de las clases de equivalencia de $\mathbb{F}_p[x]/ < f(x) >$ es un campo finito y a continuación daremos un resultado para saber cuantos elementos existen en este campo.

Teorema 1.3.1. *[23] Sea $f(x)$ un polinomio irreducible con grado m en $\mathbb{F}_p[x]$, entonces $\mathbb{F}_p[x]/ < f(x) >$ es un campo finito con p^m elementos.*

Observe que cada elemento es una clase de equivalencia como se mencionó anteriormente. Podemos comprimir la notación escribiendo los coset como un vector en $\mathbb{F}_p^m$, de la siguiente forma

$$g_{m-1}x^{m-1} + g_{m-2}x^{m-2} + \ldots + g_1x + g_0 + (f(x)) \leftrightarrow g_{m-1}g_{m-2}\cdots g_1g_0. \quad (3)$$

Esta notación permite realizar la suma en el campo finito usando la notación vectorial ordinaria.

Ejemplo 1.3.1. *El polinomio $f(x) = x^3 + x + 1$ es irreducible sobre $\mathbb{F}_2$, ya que si consideramos que es reducible entonces debería tener un factor de grado 1 y por tanto una raíz en $\mathbb{F}_2$, lo cual no pasa. Asi $\mathbb{F}_8 = \mathbb{F}_2[x]/ < f(x) >$ y usando la notación (3), los elementos de $\mathbb{F}_8$ son*

Cosets	Vectores
$0 + < f(x) >$	000
$1 + < f(x) >$	001
$x + < f(x) >$	010
$x + 1 + < f(x) >$	011
$x^2 + < f(x) >$	100
$x^2 + 1 + < f(x) >$	101
$x^2 + x + < f(x) >$	110
$x^2 + x + 1 + < f(x) >$	111

Si tomamos $x + < f(x) >$ y $x^2 + x + 1 + < f(x) >$, realizando la suma tenemos $x^2 + 1 + < f(x) >$, lo cual corresponden a la suma de $010 + 111 = 101$ en $\mathbb{F}_2^3$.

Para la multiplicación, consideremos $g_1(x) + < f(x) > \times (g_2(x) + < f(x) >$. Lo primero que debemos hacer es aplicar el algoritmo de Euclides a la multiplicación de estos elementos.

$$g_1(x)g_2(x) = f(x)h(x) + r(x). \quad (4)$$

Donde $\deg r(x) \leq m-1$ o $r(x) = 0$. Entonces

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

La notación es un poco compleja y la podemos simplificar si reemplazamos x por α y sea $f(\alpha) = 0$. Lo justificaremos en breve. De la ecuación (4), $g_1(\alpha)g_2(\alpha) = r(\alpha)$ y extendiendo la correspondencia vectorial de (3), tenemos:

$$g_{m-1}g_{m-2}\cdots g_1 g_0 \leftrightarrow g_{m-1}\alpha^{m-1} + g_{m-2}\alpha^{m-2} + \ldots + g_1\alpha + g_0. \tag{5}$$

Así la multiplicación en $\mathbb{F}_q$, es simplemente multiplicar polinomios en α de la forma ordinaria y utilizamos $f(\alpha) = 0$ para reducir las potencias de α mayores que $m-1$ a polinomios menores a m. Note que el subconjunto

$$\{0\alpha^{m-1} + 0\alpha^{m-2} + \ldots + 0\alpha + a_0 | a_0 \in \mathbb{F}_p\} = \{a_0 | a_0 \in \mathbb{F}_q\}$$

es un sub-campo de $\mathbb{F}_{q^m}$.

Ejemplo 1.3.2. *Continuando nuestro ejemplo, usando la correspondencia de la ecuación (5), nosotros obtenemos*

Vectores	Polinomios en α	Potencias de α
000	0	0
001	1	$1 = \alpha^0$
010	α	α
011	$\alpha + 1$	α^3
100	α^2	α^2
101	$\alpha^2 + 1$	α^6
110	$\alpha^2 + \alpha$	α^4
111	$\alpha^2 + \alpha + 1$	α^5

Cuadro 1.1: Potencias de α.

La columna potencias de α es obtenida usando $f(\alpha) = \alpha^3 + \alpha + 1 = 0$, lo cual implica que:

1. $\alpha^3 = \alpha + 1$.

2. $\alpha^4 = \alpha$.

3. $\alpha^3 = \alpha(\alpha + 1) = \alpha^2 + \alpha$.

4. $\alpha^5 = \alpha\alpha^4 = \alpha(\alpha^2 + \alpha) = \alpha^3 + \alpha^2 = \alpha^2 + \alpha + 1$.

Se puede continuar de la misma manera para todas las potencias de α.

Con esto terminamos nuestro primer capítulo de bases matemáticas y podemos dar paso a definir y explicar qué es la criptografía.

2 Introducción a la criptografía

Desde la antigüedad las personas han querido guardar información de forma privada y comunicarse de forma secreta entre ellos, uno de los ejemplos data de 1900 a.C en el antiguo Egipto donde fue tallada una piedra con jeroglíficos de forma cifrada [9]. Actualmente con el desarrollo tecnológico alcanzado, los sistemas de comunicación no solo nos permiten comunicarnos los unos con los otros, sino que además nos permite realizar diferentes tipos de actividades tales como el comercio electrónico, voto electrónico y autenticaciones. Si por ejemplo Alice quiere enviarle un mensaje a Bob a través de un canal inseguro (ver figura 2.1), Alice va a modificar el texto que quiere enviar con la ayuda de una clave. A este proceso lo llamamos cifrar, la idea es transformar el texto original en un texto incomprensible, el cual se denomina texto cifrado. Bob para recuperar el mensaje original debe realizar la operación inversa utilizando una clave, lo cual llamamos descifrar.

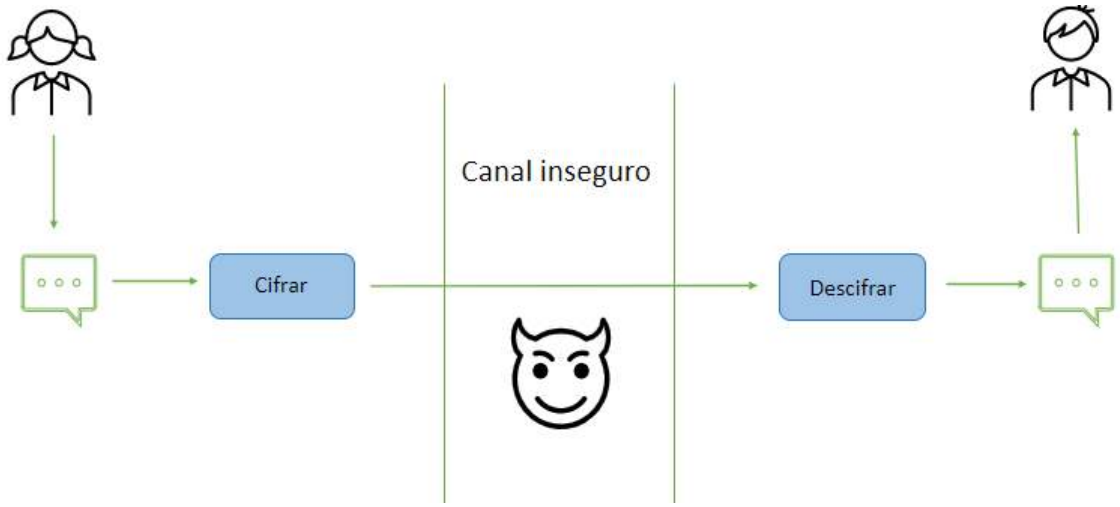

Figura 2.1: Transmisión de un mensaje a través de un canal inseguro

El área del conocimiento encargado de realizar los procesos de cifrado y descifrado se llama criptografía. El término deriva de las palabras griegas antiguas *kryptos*, que se traduce como secreto u oculto y *graphein*, lo cual significa escribir. A continuación vamos a dar unas definiciones esenciales para la criptografía:

1. **Alfabeto** $\mathscr{A}$: es el conjunto de posibles elementos utilizados para construir los mensajes a enviar.
2. **Espacio de claves** $\mathscr{K}$: es el conjunto de posibles claves (k).
3. **Algoritmo de cifrado:** un algoritmo que nos permite a partir del texto plano y una clave crear el texto cifrado.
4. **Algoritmo de descifrado:** un algoritmo que nos permite descifrar el texto cifrado recibido con la ayuda de una clave secreta para obtener el mensaje original.

Un criptosistema consta de los 4 elementos listados anteriormente y para definirlo debemos describir cada uno de ellos. La criptografía es la encargada de la creación y el diseño de los criptosistemas y el criptoanálisis se encarga de analizar su seguridad.

Este capítulo tiene como objetivo introducir la criptografía, dar sus principales definiciones y su evolución a lo largo del tiempo.

2.1. Criptografía simétrica

En la criptografía simétrica, la clave de cifrado y descifrado es la misma y debe permanecer secreta. Alice y Bob deben encontrar un canal seguro a través del cual puedan entregarse la clave secreta (ver figura 2.2).

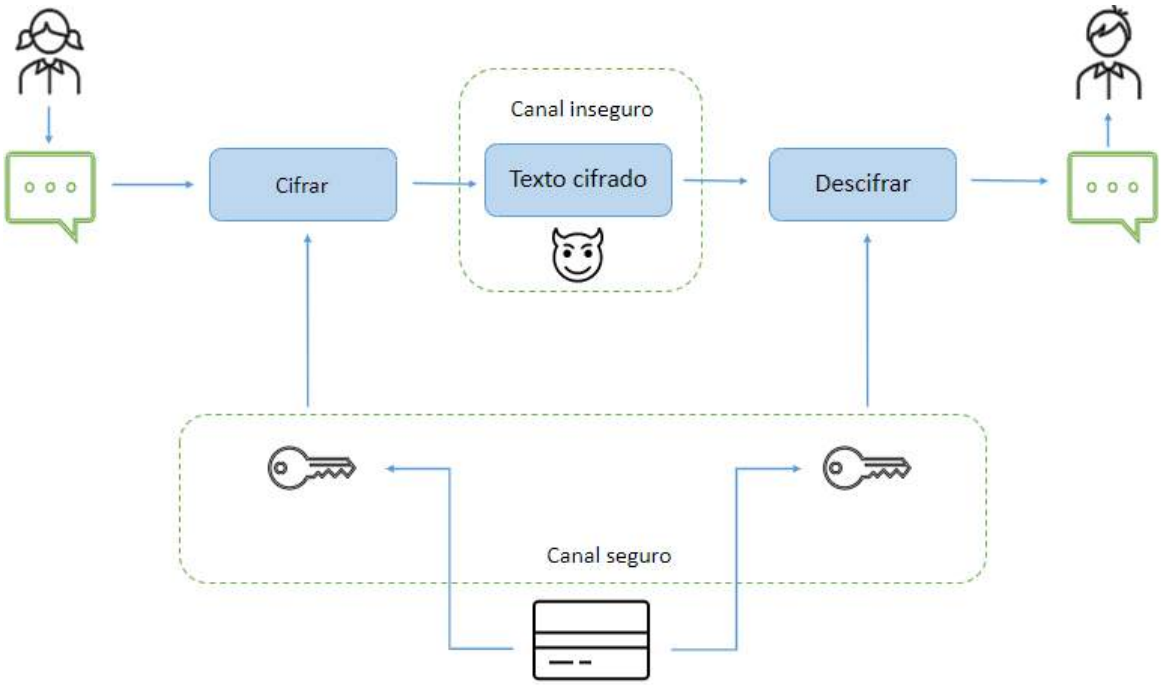

Figura 2.2: Criptografía Simétrica.

Ejemplo: Cifrado de César

El cifrado de César es uno de los ejemplos más famosos que existen de criptografía simetría. Es nombrado en honor al emperador Julio César, quien lo utilizaba para cifrar sus mensajes y poder comunicarse con sus amigos y aliados políticos. En este cifrado utilizamos el alfabeto latino y a cada letra le asociamos un número, como se muestra en la Figura 2.3.

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

Figura 2.3: Abecedario

El conjunto de claves secretas corresponde a $\mathscr{K} = \{k \in \mathbb{Z} : 0 < k < 26\}$. Para cifrar el mensaje $m = (m_1, m_2, \cdots, m_n)$, debemos primero convertir cada una de sus letras m_i es su correspondiente número dada la Figura 2.3 anterior. De esta forma se obtiene un mensaje $m' = (m'_1, m'_2, \cdots, m'_n)$ compuesto de números y asumiendo

que k es la clave secreta, calculamos:

$$c'_i = e(m'_i) = (m'_i + k) \mod 26$$

y obtenemos $c' = (c'_1, c'_2, \cdots, c'_n)$. Por último se sustituye cada uno de los c'_i utilizando la Figura 2.3 obtenemos el mensaje $c = (c_1, c_2, \ldots c_n)$ de letras el cual es enviado por el canal inseguro. Para el descifrado repetimos los pasos anteriores solo que ahora vamos a utilizar la transformación $d(c) = (c - k) \mod 26$.

Por ejemplo, utilizando $k = 3$, si Alice le quiere enviar un mensaje a Bob dándole a conocer el lugar de su siguiente reunión. Alice toma el mensaje (BOGOTÁ) e inicia el cifrado tomando la letra $B = 1$ que será reemplazada por la letra $1 + 3 \mod 26$ que corresponde a la letra E. Note que esto equivale a correr la letra 3 posiciones a la derecha en el alfabeto, tal como se muestra en la Figura 2.4. Al repetir este proceso Alice obtiene el mensaje cifrado ERJRWD, el cual puede ser enviado a Bob por un canal inseguro. Una vez Bob reciba este mensaje, deberá realizar el procedimiento contrario, es decir, tomar la letra E y desplazarla 3 posiciones hacia la izquierda para obtener la letras B como se puede ver en la Figura 2.4.

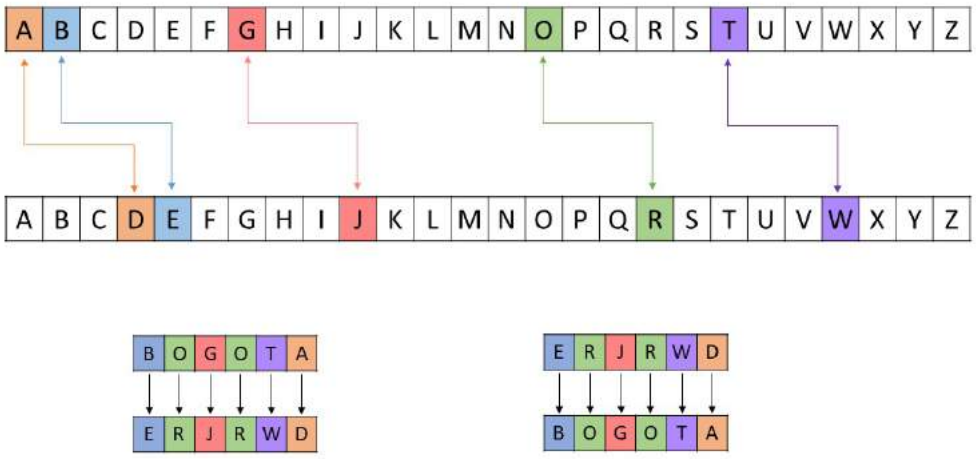

Figura 2.4: Ejemplo de Cifrado de César.

El desarrollo de las máquinas llevado a cabo a principios del siglo XX, impactó las comunicaciones que se llevaban hasta ese momento y en consecuencia genera un cambio en la criptografía, no solamente por la capacidad computacional sino también en la forma en como se realizan los algoritmos de cifrado y descifrado. Algunos ejemplos, son las máquinas ENIGMA y PURPLE las cuales fueron utilizadas en la segunda guerra mundial por Alemania y Japón, respectivamente, para cifrar y descifrar sus mensajes. El trabajo de Alan Turing para descifrar la máquina ENIGMA definió el rumbo en la segunda guerra mundial.

La llegada de los computadores genera otro cambio en las comunicaciones y en la criptografía. Por un lado, tenemos la velocidad de cómputo y los tipos de algoritmos que se pueden desarrollar; por otro lado surge un cambio conceptual en el uso del alfabeto, debido a que no se usa el abecedario tal como se vio en el cifrado de César, sino que ahora utilizamos una cadena de bits y por lo tanto, el alfabeto empleado para crear los mensajes son números. Esto dio paso a que la criptografía se apartara de los patrones lingüísticos y lexicográficos e hiciera uso de las matemáticas y las ciencias de la computación.

En efecto, los bits son la unidad de información básica que utiliza los computadores y son representados por unos y ceros. Cada vez que escribimos en un computador un caracter, se convierte en un conjunto de unos y ceros siguiendo un determinado código. Los avances tecnológicos mostraron la necesidad de desarrollar estándares de comunicación. En el año 1963 se propone el American Standard Code for information Interchange (ASCII), el cual nos permite representar cada uno de los diversos caracteres con 8 bits.

En la criptografía simétrica moderna existen dos principales familias, el cifrado de flujo y el cifrado por bloque.

En el **cifrado de flujo** el alfabeto es $\mathscr{A} = \{0,1\}^*$, donde $*$ es un número indefinido, es decir, los mensajes m, son una cadena de bits de tamaño indefinido. El conjunto $\mathscr{K}$, es generado de forma pseudoaleatoria y sus elementos son cadenas de bits del

mismo tamaño del mensaje. El cifrado del mensaje m se realiza haciendo un XOR bit a bit con la clave secreta k. La operación XOR se simboliza por $\oplus$ y representa una suma módulo dos (ver Tabla 2.1). El texto cifrado es entonces $e(m) = (m_1 \oplus k_1, m_2 \oplus k_2, \ldots)$.

x	y	x $\oplus$ y
0	0	0
0	1	1
1	0	1
1	1	0

Cuadro 2.1: Operaciones XOR.

Para descifrar el mensaje c se debe aplicar la misma operación XOR con la clave secreta para recuperar el mensaje original m. Una de las ventajas del cifrado por flujo es la velocidad con la que se pueden realizar los algoritmos de cifrado y descifrado, debido a que existen algoritmos potentes que realizan este procedimiento. Otra ventaja se presenta cuando existe un error en un bit del mensaje cifrado, ya que solo afecta a este bit y no se propaga por todo el mensaje. Sin embargo, podemos observar que toda la información del mensaje se está encapsulando en un único texto cifrado, lo cual no siempre es bueno. Además, otro problema surge cuando intentamos hacer un envío constante de información ya que tendríamos que estar cifrando mensajes muy largos y esto es muy costoso computacionalmente. En el **cifrado por bloque**, ya no se cifra bit a bit el mensaje m, sino que hacemos una división por bloques de bits de tamaño fijo, los cuales son cifrados por el algoritmo para luego ser enviados. La característica de este cifrado está dada por el tamaño del bloque y el tamaño de la clave secreta.

Por ejemplo, en 1970 Horst Feistel propuso un cifrado por bloque, representado en la figura 2.5. El cual fue la base para uno de los criptosistemas simétricos por bloques más conocidos (DES). Un punto importante del cifrado de Feistel es el hecho que tiene diferentes rondas, a continuación, vamos a explicar una de ellas. Comenzamos dividiendo el mensaje m en dos submensajes del mismo tamaño, la parte izquierda I y la parte derecha D. Luego estos submensajes son la entrada para una

función no lineal F y finalmente hacemos un XOR con el mensaje I del cual obtenemos I', luego se intercambian los mensajes I' y D. Finalmente repetimos este procedimiento un cierto número de iteraciones, tal como se ve en la Figura 2.5. Debemos tener en cuenta que dependiendo del criptosistema que se quiere emplear se deberá definir la función F y el generador de las claves secretas.

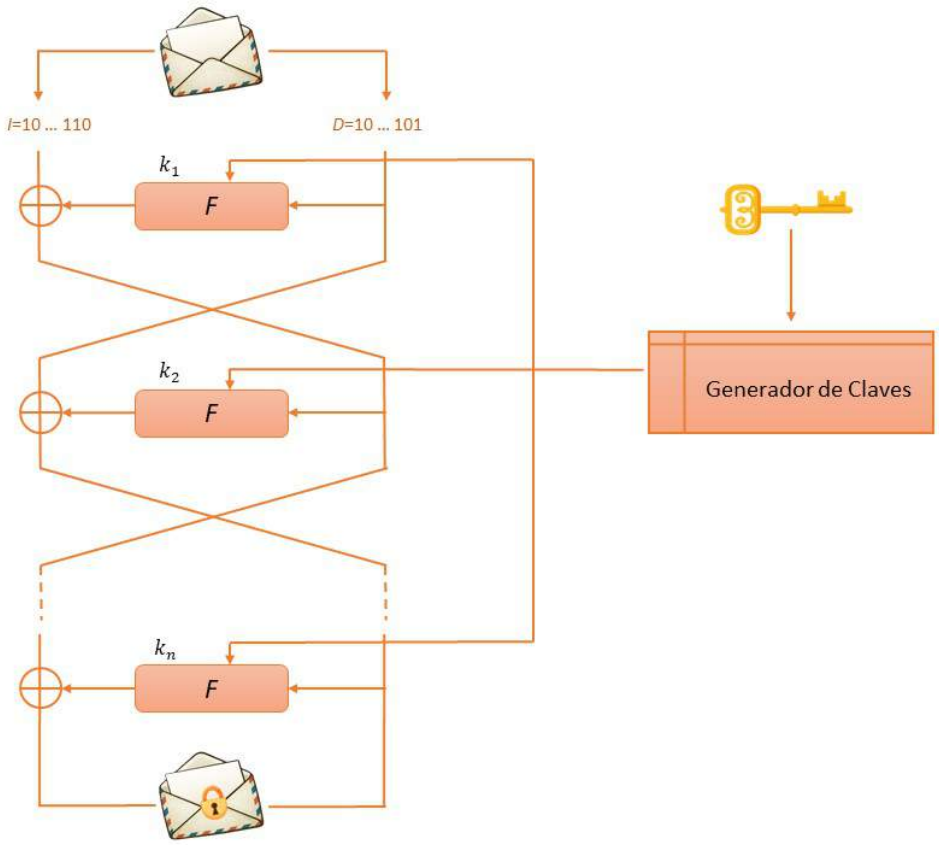

Figura 2.5: Criptosistema de Feistel

En criptografía es muy importante poder encontrar un criptosistema estándar para poder comunicarnos de forma segura entre diferentes entidades. En 1973, la agencia nacional de estándares (NSA), hizo una convocatoria para elegir el criptosistema estándar y usarlo para cifrar datos gubernamentales confidenciales. El ganador fue el cifrado de Feistel con algunas modificaciones para cumplir con la norma federal para el procesamiento de información. El 15 de julio de 1977 se nombró el estándar federal de cifrado de datos o DES (*Data Encryption Standard*). Este mismo año Dif-

fie y Hellman proponen una máquina especial que encuentra una clave que podría romper el DES. En consecuencia a este ataque y otros nuevos que surgen años después [45], la NIST propone otro concurso en 1997. Finalmente en el 2000 se anuncia que el algoritmo de Rijndael, propuesto por Vincent Rijmen y Joan Daemen, es el nuevo estándar, convirtiéndose en el esquema de cifrado líder en el mundo en ese momento. Este estándar consiste en procesar bloques de 128 bits utilizando diferentes tipos de claves secretas con un tamaño de 128, 192 o 256 bits.

La criptografía a clave simétrica juega actualmente un papel muy importante. El AES es uno de los criptosistemas más utilizados. En el modelo de criptografía que hemos estado estudiando hasta ahora, Alice y Bob eligen en secreto una clave para poder intercambiar información. Hasta el momento esta clave se usa tanto para el cifrado como para el descifrado, un ejemplo es el DES. Un inconveniente de los criptosistemas de clave simétrica es la previa comunicación de la clave secreta entre Alice y Bob, ya que se debe utilizar un canal seguro para entregarse y así poder transmitir cualquier información. En la práctica, esto puede resultar muy difícil de lograr. Por ejemplo, supongamos que Alice y Bob viven en lugares distantes uno del otro y deciden que quieren comunicarse electrónicamente. En una situación como esta, es posible que Alice y Bob no tengan acceso a un canal seguro razonable.

Adicionalmente, si Alice quiere comunicarse con Carlos y utiliza la misma clave secreta establecida con Bob, esto implicaría que Bob tendría acceso a toda la información, lo cual no obligatoriamente es deseable. Alice entonces debería generar otra clave secreta para comunicarse con Carlos y en general Alice debe producir una clave secreta por cada persona con la que se quiera comunicar. La criptografía asimétrica o a clave pública que veremos a continuación, nos proponen criptosistemas que solucionan estos dos inconvenientes.

2.2. Criptografía a clave pública

La criptografía asimétrica, también conocida como criptografía a clave pública es un esquema en el que se utilizan dos claves distintas. Una clave pública, utilizada para el cifrado que se pone a disposición del público; y una clave privada utilizada para el descifrado que siempre debe permanecer secreta. En criptografía a clave pública, si Alice quiere enviar un mensaje a Bob, va a cifrar el mensaje utilizando la clave pública de Bob, el resultado es enviado por un canal inseguro y Bob es la única persona que lo puede descifrar, debido a que tiene la clave secreta (ver Figura 2.6). Este tipo de cifrados se puede ver como una función de un solo sentido, ya que cifrar es rápido y se implementa utilizando una clave pública que cualquiera conoce. Sin embargo, para el descifrado no se puede si no se tiene conocimiento de una clave secreta.

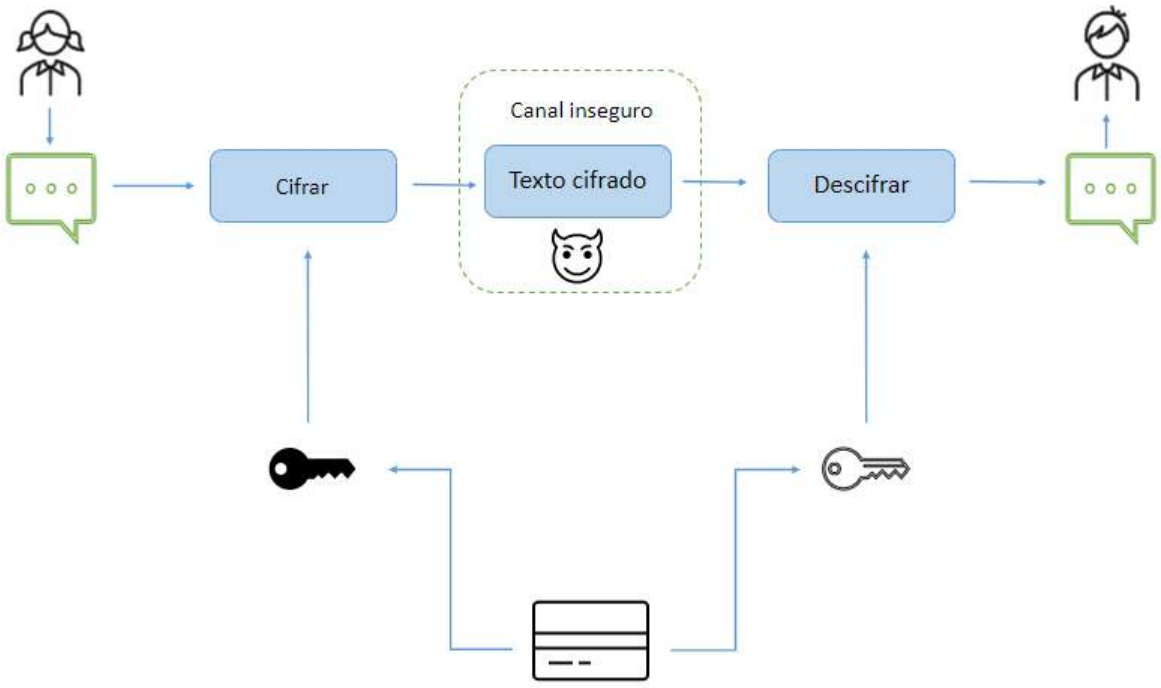

Figura 2.6: Criptografía a clave pública.

El primer criptosistema a clave pública llamado RSA fue publicado en 1977 por Ron Rivest, Adi Shamir y Leonard Adleman. Su seguridad se basa en los problemas de factorización de números primos grandes y el logaritmo discreto. La seguridad del

cifrado está directamente relacionada con el tamaño de la clave. Si el tamaño se duplica, la seguridad del algoritmo aumenta exponencialmente. Las claves del RSA típicas son de 2048 o 4096 bits.

Ejemplo: RSA

Primero Alice debe seleccionar dos números primos grandes p y q. Ella deberá mantener estos números en secreto. Entonces Alice calcula el producto $n = p \times q$. Luego ella calcula la función de Euler para n, $\phi(n) = (p-1)(q-1)$, lo cual representa el número de primos relativos menores que n. Para un número primo p hay $p-1$ números primos relativos a él, porque no tiene otros factores más que 1 y él mismo. Alice entonces selecciona un número e donde $1 < e < \phi(n)$ y tal que $\gcd(e, \phi(n)) = 1$, es decir e y $\phi(n)$ son primos relativos. Entonces e tiene un inverso multiplicativo modulo $\phi(n)$, el cual llamaremos d. Esto significa que $e \times d = 1($ mód $\phi(n))$.

Alice debe calcular a continuación d. Esto se puede hacer fácilmente usando el algoritmo extendido de Euclides. Alice tiene ahora una clave pública (e, n) y una clave privada (p, q, d). Bob ha realizado los mismos pasos y tiene su propio par de claves públicas y privadas. Alice da a conocer su clave pública en un directorio que está disponible para cualquiera que quiera enviarle un mensaje.

Para cifrar un mensaje usando el RSA, Bob primero debe convertir el mensaje m en un número. Esto suele ser fácil debido a que se reemplaza cada letra de m por su número correspondiente en el código ASCII, como lo hemos visto en la sección anterior. A continuación, Bob toma la clave pública de Alice (e, n) y calcula $c = m^e$ mód n. El valor de c es el mensaje cifrado, el cual se envía por el canal inseguro. Para descifrar el mensaje de Bob, se debe usar la clave secreta de Alice (p, q, d) y por último se calcula $m = c^d$ mód n para recuperar el mensaje.

Por ejemplo, si Bob quiere comunicarse con Alice utilizando el RSA, él debe realizar los siguientes pasos:

1. **Generación de claves:** Primero Alice debe generar sus claves, para ello selec-

ciona $p = 17$ y $q = 11$, luego calcula $n = p \times q = 17 \times 11 = 187$ y la función $\phi(n) = (p-1) \times (q-1) = 16 \times 10 = 160$. Luego, Alice debe escoger un valor para e tal que el $gcd(e, 160) = 1$, ella opta por $e = 7$. Lo siguiente que debe hacer es calcular el valor de d, para este caso tenemos que $e \times d = 1 \mod 160$ y $d < 160$, por lo que $d = 23$ porqué $23 \times 7 = 161 = 1 \mod 160$. Finalmente Alice da a conocer su clave pública $(7, 187)$ y mantiene en secreto $(17, 11, 23)$.

2. **Cifrar y descifrar:** Supongamos que Bob necesita enviarle a Alice el mensaje $m = 88$, entonces el debe tomar la clave pública de Alice $(7, 187)$ y calcular $c = 88^{77} \mod 187 = 11$. Cuando Alice reciba el mensaje c, ella debe utilizar su clave secreta $(17, 11, 23)$ y calcular $m = 11^{23} \mod (17 \times 11) = 88$.

Mas de 40 años después del nacimiento del RSA, nuestra criptografía a clave pública se basa en dos problemas: factorización de enteros y el logaritmo discreto. La criptografía asimétrica es indispensable para la seguridad de redes informáticas, en particular internet. Por ejemplo, cada día miles de millones de descargas de software y conexiones de comunicación están protegidos por firmas digitales y criptosistemas de clave pública. Para claves del RSA muy largas, la cantidad de tiempo que se necesitaría para romperse seria astronómico. Pero, ¿y si no fuera así? ¿Qué pasaría si pudiéramos construir una computadora que pudiera resolver el RSA en segundos en lugar de milenios? ¿Estaría muerta la criptografía? ¿Colapsaría el comercio de Internet? Como hemos visto en el capítulo anterior Peter Short propone un algoritmo que soluciona los problemas de la factorización de números y el logaritmo discreto utilizando un computador cuántico ¿Cómo pudo pasar esto? Esa es la perspectiva y el peligro de la computación cuántica.

2.3. Criptografía post-cuántica

En 1994 Peter Shor descubrió un algoritmo cuántico de tiempo polinomial que resuelve el problema de factorización de primos grandes y del logaritmo discreto en la

criptografía a clave pública. Por lo tanto, todos estos criptosistemas que se utilizan actualmente en la práctica se volverán inseguros una vez que se puedan construir ordenadores cuánticos suficientemente grandes. Debido a la relevancia de la criptografía a clave pública y la amenaza cada vez mas realista de estos computadores cuánticos, fue necesario idear el concepto de criptografía post-cuántica, la cual tiene como propósito diseñar criptosistemas que logren resistir los ataques de esta nueva tecnología. La NIST ha creado una nueva competencia para encontrar algoritmos post-cuánticos que han atraído a 65 participantes de todo el mundo.

Como aún se desconocen los detalles técnicos sobre los computadores cuánticos, no es posible un análisis más detallado de la seguridad post-cuántica. Actualmente esta seguridad hace referencia a un esquema criptográfico asociado a un problema computacional que no tenga solución en tiempo polinomial en una computadora cuántica. Esto implica la imposibilidad de resolverlo en una computadora convencional.

Sin embargo, la buena noticia es que ya existen sistemas criptográficos que ofrecen protección contra potentes computadores cuánticos. Actualmente hay algoritmos criptográficos basados en problemas matemáticos que no pueden resolverse fácilmente con una computadora cuántica, estos algoritmos se dividen en cinco categorías:

- Funciones de Hash.
- Teoría de códigos.
- Laticces.
- Álgebra multivariada.
- *Isogenys* supersingular.

Cada categoría se centra en un conjunto diferente de problemas matemáticos, algunos de ellos tan antiguos y bien entendidos como las matemáticas de la criptografía a clave pública actual. En esta tesis nos vamos a enfocar en el área de la teoría de

códigos para así poder comprender por qué esta ciencia nos brinda una salida al problema de la seguridad con la llegada de los computadores cuánticos. A continuación veremos una introducción a la computación cuántica.

3 | Computación Cuántica

Un computador cuántico es un computador basado en los principios de la física cuántica y puede ser entendido usando un modelo de computación basados en estos principios. Esta ciencia es un subcampo de la física que estudia los fenómenos a nivel microscópico, a muy bajas temperaturas o que se encuentran muy aisladas del ambiente que los rodea. El área relacionada con los computadores cuánticos se denomina generalmente computación cuántica, que se considera un campo prometedor desde el punto de vista científico e informático.

Los computadores cuánticos representan la información al hacer uso de fenómenos sobre átomos a nivel microscópico y emplean sistemas mecánicos cuánticos con algunas propiedades para realizar cálculos. En consecuencia, los computadores cuánticos tienen hardware y un mecanismo computacional que son radicalmente diferentes de los computadores clásicos, es decir tanto el hardware como el software se construyen sobre la base de los principios de la mecánica cuántica. Este es uno de los puntos más importantes para comprender los computadores cuánticos.
Los computadores actuales representan información por medio de un bit que solo puede tener dos valores 0 o 1. La unidad básica de información en los computadores cuánticos se les conoce como qubit. Se basa en estados de superposición en los que tanto el estado 0 como el 1 se superponen. Esto implica que lo qbits puedan representar más información que los bits. Por lo tanto, si se dan varios qubits los computadores cuánticos permiten el cálculo a una velocidad mayor. Estas diferencias permiten que los computadores cuánticos puedan realizar cálculos de problemas cuya soluciones se consideran imposibles con los computadores tradicionales.

Sin embargo, surge la pregunta ¿Qué otros problemas pueden resolverse utilizando los computadores cuánticos? La respuesta corta sería no se sabe. Crear algoritmos cuánticos puede llegar a ser difícil, debido a que los diseñadores de estos algoritmos pueden enfrentarse a dos problemas difíciles, los cuales no se presentan en el diseño de algoritmos para computadores clásicos. Primero, la intuición humana tiene sus raíces en el mundo clásico. Por lo que si utilizamos esta intuición para diseñar algoritmos entonces obtendremos algoritmos que podrán solucionar problemas en el mundo clásico. Entonces para diseñar algoritmos cuánticos, se debe tener en cuenta que la intuición clásica está presente pero no siempre se puede confiar en ella. En segundo lugar, para ser realmente interesante no basta con diseñar un algoritmo cuántico que funcione con todos los principios de la mecánica cuántica, si no ser excelente en términos de rendimiento en comparación con cualquier algoritmo clásico. Entonces, es posible que se pueda encontrar un algoritmo que haga uso de las propiedades de la mecánica cuántica, que sin embargo no es de interés generalizado porque existen algoritmos clásicos que tienen mejor complejidad computacional. La combinación de estos dos problemas nos dará como resultado algoritmos cuánticos desafiantes para el futuro.

Ahora, explicaremos un poco de historia sobre la computación cuántica. Mucha gente ha tenido la oportunidad de aportar ideas en esta ciencia y formalizar sus modelos de computación.

Comencemos en 1980, Paul Benioff señaló la posibilidad de construir una computador basado en la mecánica cuántica [1], propuso un modelo mecánico cuántico de computadores dentro del marco de maquinas de Turing, que es un modelo estándar de computación. El punto de partida de Benioff fue hacer circuitos lógicos más pequeños, este modelo se puede interpretar como un modelo de computación que satisface las leyes de la mecánica cuántica. Benioff sugirió usar los diferentes giros de partículas elementales para representar dos dígitos binarios. Lo cual significa que los cálculos se realizan de forma cuántica sin consumo de energía. Sin embargo, el modelo que propuso Benioff es equivalente a una maquina de Turing y por

ende no podemos superar el poder de una computadora clásica. En consecuencia no podemos interpretar este modelo para los computadores cuánticos.

Siguiendo con la línea del tiempo tenemos que, en 1965 Richard Feynman (Premio Nobel de Física del año 1965) habló de un computador cuántico que puede simular la física [16]. Aunque las contribuciones de Feynman a la computación cuántica son muy conocidas, su principal artículo señaló que algunos fenómenos de la mecánica cuántica no pueden ser simulados de manera efectiva por los computadores clásicos y afirmó que la computación necesaria para simular esta física se puede realizar de forma más eficiente utilizando fenómenos de la mecánica cuántica. Finalmente, Feynman presentó algunos ejemplos de algunos fenómenos físicos que pueden interpretarse por medio de un modelo de computación cuántico.

En 1985 el físico inglés David Deutsch propuso un modelo de computación llamado computador cuántico el cual replantea una máquina de Turing clásica [11]. El computador cuántico de Deutsch es el primer modelo de computación para la cuántica. Debido a que su teoría es más concreta que la de Feyman. Además Deutsch también mostró una teoría de compuertas cuánticas, que es otro modelo de computadoras cuánticas, en 1988 [12]. Una compuerta cuántica es una compuerta lógica para las computadoras cuánticas.

En 1994 surge una propuesta la cual es muy relevante en esta tesis. Peter Shor mostró un algoritmo cuántico de tiempo de ejecución polinomial sobre el número N el cual soluciona los problemas de la factorización de números primos y logaritmo discreto [40]. Este resultado trae una consecuencia fuerte en el diseño de computadores cuánticos. Esto se debe a que la factorización de números primos se conoce como un problema complicado de solucionar en teoría de la computación clásica. En consecuencia se ve reflejada la importancia de los computadores cuánticos y no solo por poder resolver problemas que no sabemos como resolver en computadores clásicos, sino por que tienen un impacto en criptografía, cómo vimos al inicio del Capítulo 2.

Entendamos cómo funciona la computación cuántica. Al principio de este capítulo, vimos que la unidad básica de procesamiento de información en un computador moderno es el bit, que puede tomar uno de los dos estados que etiquetamos como 0 y 1. De manera análoga, podemos definir una unidad básica de procesamiento de información que se puede utilizar en computación cuántica.

En el caso de un qubit, etiquetamos estos dos estados con $|0\rangle$ y $|1\rangle$. En la teoría cuántica, un objeto encerrado usando la notación $|\cdot\rangle$ puede llamarse estado, vector o ket.

Entonces, ¿en qué se diferencia un qubit de un bit ordinario? Mientras que un bit en una computador ordinario puede estar en el estado 0 o en el estado 1, un qubit es algo más general. Un qubit puede existir en el estado $|0\rangle$ o en el estado $|1\rangle$, pero también puede existir en lo que llamamos un estado de superposición. Este es un estado que es una combinación lineal de los estados $|0\rangle$ y $|1\rangle$. Llamemos este estado $|\psi\rangle$, la superposición se escribe como:

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle \tag{1}$$

donde α, β son números complejos.

Cuando se mide un qubit, solo se encontrará en el estado $|0\rangle$ o en el estado $|1\rangle$. Las leyes de la mecánica cuántica nos dicen que el módulo al cuadrado de α y β en la ecuación (1) nos da la probabilidad de encontrar el qubit en el estado $|0\rangle$ o $|1\rangle$ respectivamente. En otras palabras:

1. $|\alpha|^2$: Es la probabilidad de encontrar ψ en el estado $|0\rangle$.
2. $|\beta|^2$: Es la probabilidad de encontrar ψ en el estado $|1\rangle$.

Dado que los cuadrados de estos coeficientes son la probabilidad de obtener un resultado de medición, α y β están restringidos por el siguiente requisito:

$$|\alpha|^2 + |\beta|^2 = 1$$

A continuación, describimos brevemente el algoritmo de Shor. Tenga en cuenta que para comprender el algoritmo de Shor, necesitamos algunos antecedentes de la teoría de números y la transformada de Fourier.

La idea de Shor es reemplazar el problema de la factorización por el problema de la búsqueda de períodos y resolverlo mediante la transformada cuántica de Fourier. El estudio de Shor se vio influenciado por el algoritmo de Simon que encontró la periodicidad de una función, que se propuso aproximadamente al mismo tiempo [44]. Shor utilizó el método de factorización haciendo uso de la propiedad de que los residuos tienen cierta periodicidad.

Sean a y b números enteros. Denotamos su máximo común divisor por $gcd(a, b)$. El algoritmo para encontrar la factorización de un entero N se puede escribir de la siguiente manera:

Algoritmo de factorización

1. Seleccionar un entero x adecuado tal que $x < N$.
2. Calcular $f = gcd(x, N)$. Si $f \neq 1$, entonces f es un factor de N. De lo contrario continuar al siguiente paso.
3. Encuentre el menor entero r tal que su residuo sea 1 al dividir x^r por N, *i.e.*, $x^r \mod N = 1$. Esta es la parte en donde Shor aplica elementos de la computación cuántica como lo es la transformada de Fourier, compuertas cuánticas entre otros.
4. Si $gcd(x^{\frac{r}{2}} - 1, N)$ o $gcd(x^{\frac{r}{2}} + 1, N)$ no son 1, entonces son los factores de N. De lo contrario volver al item (3).

4 | Códigos lineales

En 1948 Claude Shannon publicó el articulo *A Mathematical theory of communication* [39], el cual supone el comienzo de la teoría de códigos. Esta teoría tiene como objetivo encontrar métodos óptimos y confiables que corrijan de forma eficiente los posibles errores que se hayan generado por el ruido del canal utilizado en el envío. De ahora en adelante, supondremos que los errores generados por el canal tienen las siguientes características: cada error ocurre al azar, de forma independiente y el número de errores es pequeño comparado con el tamaño del mensaje.

Un método simple podría ser enviar la información varias veces. Por ejemplo, supongamos que Alice le quiere enviar a Bob un mensaje $u = (u_0, u_1, u_2)$. Alice y Bob hablan antes y acuerdan que Alice enviará cada símbolo 3 veces, es decir que Bob debería recibir $(u_0, u_0, u_0, u_1, u_1, u_1, u_2, u_2, u_2)$. De esta manera, Bob tiene más posibilidades de identificar posibles errores y recuperar el mensaje original. Si por ejemplo Bob recibe $(u_0, a, u_0, u_1, u_1, u_1, u_2, u_2, b)$, el puede detectar que los errores están en las posiciones 2 y 9 y así saber que el mensaje u es (u_0, u_1, u_2). El método mencionado anteriormente es un buen ejemplo y funciona debido a que el mensaje recibido contiene pocos errores, sin embargo no es eficiente ya que el tamaño del mensaje se multiplica por 3, lo cual eleva el costo computacional.

En teoría de códigos, la idea es modificar el mensaje original que se quiere enviar (u de tamaño k) agregándole una información adicional, obteniendo así lo que llamamos la palabra código (c de tamaño n), donde $n > k$. A este proceso lo denominamos codificar. Lo que recibe Bob es el mensaje $y = c \oplus e$, donde e es el vector del

error agregado por el canal y la mayoría de sus elementos son ceros. La operación $\oplus$ hace referencia a la suma bit a bit de cada componente de c y e. Llamamos decodificar al algoritmo que puede corregir el vector e y recuperar los vectores c o u, dependiendo de la literatura. Esto se puede ver en la figura 4.1.

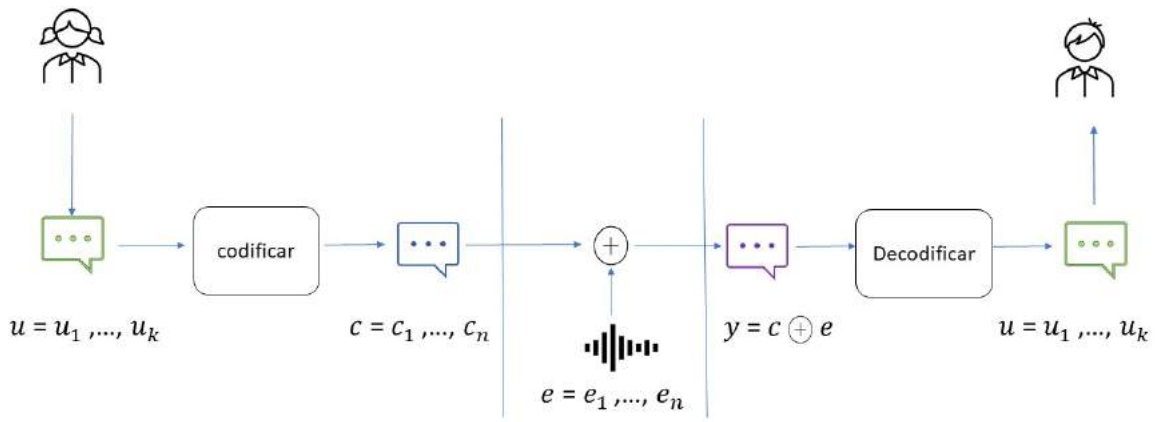

Figura 4.1: Comunicación usando corrección de códigos.

En este capítulo estudiaremos los conceptos básicos de la teoría de códigos y presentaremos algunos códigos necesarios para esta tesis. Nos limitaremos al estudio de los códigos lineales.

4.1. Conceptos básicos de la teoría de códigos

Los códigos lineales son códigos de bloque ya que para codificar un mensaje se debe partir en bloques de tamaño fijo. De ahora en adelante el tamaño de cada uno de los bloques lo denotaremos como k.

Definición 4.1.1 (Código de bloques). *[23] Sea $\mathcal{A}$ un alfabeto finito con q símbolos. Un código de bloques con tamaño n que contiene m palabras código sobre el alfabeto $\mathcal{A}$, es el conjunto definido como $\mathscr{C} = \{c_1, c_2, \ldots, c_m\}$ tales que $c_i = (c_{i0}, c_{i1}, \ldots, c_{in-1})$ donde los*

$c_{ij} \in \mathcal{A}$ *para todo* $j \in [0, n-1]$. *A este tipo de códigos los denominamos* $\mathscr{C}(n,k)$.

La relación entre k y n es una medida que nos interesa analizar y corresponde a la fracción de información que no es redundante en la palabra código.

Definición 4.1.2. *[23] La tasa en un código* $\mathscr{C}(n,k)$ *está definida por* $R = \frac{k}{n}$.

Supongamos que $\mathcal{A} = \mathbb{F}$, un campo finito con q elementos. El conjunto de todos los posibles mensajes que podemos transmitir, es el conjunto de tuplas con k elementos en $\mathbb{F}$. Sabemos entonces que tenemos q^k mensajes distintos.

Recordemos que un espacio vectorial es un conjunto dotado por las operaciones suma y multiplicación por escalar. Una base del espacio vectorial es un conjunto maximal que contiene vectores linealmente independientes, tales que cualquier elemento que pertenezca a este conjunto se puede expresar como una combinación lineal de estos elementos. Llamamos **dimensión** del espacio vectorial a la cantidad de elementos que contiene la base. Teniendo esta idea presente, introducimos nuestra definición de un código lineal. Estos códigos tienen una estructura algebraica que nos permite encontrar algoritmos para codificar y decodificar de forma eficiente.

Definición 4.1.3. *[23] Un código lineal* $\mathscr{C}(n,k)$ *es un subespacio vectorial de* $\mathbb{F}^n$ *de dimensión k.*

El código es llamado lineal ya que si $\mathscr{C}$ es un subespacio de $\mathbb{F}^n$, entonces para todo $c_i, c_j \in \mathscr{C}$ tenemos que $c_i + c_j \in \mathscr{C}$. Adicionalmente para todo $c_i \in \mathscr{C}$ y $f \in \mathbb{F}$, tenemos que $c_i f \in \mathscr{C}$.

Como $\mathscr{C}$ es un subespacio vectorial, sabemos que en particular el vector de ceros está en $\mathscr{C}$ y por lo tanto cero es siempre es una palabra código.

A continuación vamos a definir el concepto de distancia de Hamming, la cual utilizaremos a lo largo de esta tesis.

Definición 4.1.4. *[23] La distancia de Hamming* $d(x,y)$ *entre dos palabras código* x *y* y *es*

el número de coordenadas en las cuales difieren, es decir

$$d(x,y) = |\{1 \leq i \leq k | x_i \neq y_i\}|.$$

Teorema 4.1.1. *La distancia de Hamming es una métrica sobre el conjunto de palabras códigos.*

Demostración. Tenemos que demostrar que la función d satisface los axiomas de espacio métrico, los cuales son:

1. $d(x,y) \geq 0$
2. $d(x,y) = 0$ si y solo si $x = y$
3. $d(x,y) = d(y,x)$
4. $d(x,z) \leq d(x,y) + d(y,z)$

Para ello escojamos dos palabras código

$$x = (x_1, x_2, \ldots, x_n) \in \mathscr{C}$$

$$y = (y_1, y_2, \ldots, y_n) \in \mathscr{C}.$$

Por la definición

$$d(x,y) = \sum_{i=1}^{n} f(x_i, y_i)$$

donde

$$f(x_i, y_i) = \begin{cases} 1 \quad si \quad x_i \neq y_i \\ \\ 0 \quad si \quad x_i = y_i \end{cases}$$

Observe que cada término de la suma es no negativo. Por lo tanto (1) se cumple. Adicionalmente, la suma es igual a cero si y solo si $f(x_i, y_i) = 0$ para todo i, es decir, si $x = y$ y por lo tanto (2) se cumple.

Continuamos con la simetría, esta propiedad se demuestra inmediatamente ya que para cada i, tenemos $f(x_i, y_i) = f(y_i, x_i)$. Ahora para la desigualdad triangular, sea $z = (z_1, z_2, \ldots, z_n) \in \mathscr{C}$. Entonces

$$d(x, z) = \sum_{i=1}^{n} f(x_i, z_i).$$

Ahora para cada i

$$f(x_i, z_i) \leq f(x_i, y_i) + f(y_i, z_i).$$

Observe que tenemos los siguientes casos:

1. Si $x_i = z_i$, entonces los posibles valores que toma $f(x_i, y_i) + f(y_i, z_i)$ son $0, 2$ y por tanto la desigualdad se cumple.

2. Si $x_i \neq y_i$, luego los posibles valores para $f(x_i, y_i) + f(y_i, z_i)$ son $0, 1, 2$. Observe que para los valores $1, 2$ la desigualdad se mantiene. Sin embargo, no es posibles que esta suma sea cero, ya que si lo fuera tendríamos que $x_i = y_i = z_i$, lo cual es una contradicción.

En consecuencia tenemos que la distancia de Hamming induce una métrica sobre $\mathscr{C}$. □

Definición 4.1.5 (Distancia mínima). *[46] Sea $\mathscr{C}(n, k)$ un código y $d(x, y)$ la distancia de Hamming. La distancia mínima de $\mathscr{C}$ está definida por*

$$d = min\{d(x, y) : x, y \in \mathscr{C}, x \neq y\}.$$

En otras palabras, la distancia mínima de un código es la mínima distancia entre dos palabras código distintas. La distancia mínima d es un parámetro muy importante para el código y por tanto añadimos esta información al definirlo: $\mathscr{C}(n, k, d)$.

Otro concepto importante para un código lineal $\mathscr{C}(n, k, d)$ es el peso de cada palabra código, lo llamaremos *peso de Hamming* y se denotara como $wt(x)$.

Definición 4.1.6 (Peso de Hamming). *[46] El peso de Hamming de un vector $x \in \mathbb{F}^n$, denotado por $wt(x)$, es el número de coordenadas distintas de cero. Es decir:*

$$wt(x) = d(x, 0)$$

donde 0 *es el vector cero.*

Definición 4.1.7 (Peso mínimo). *El peso mínimo de un código $\mathscr{C}(n, k, d)$, es el menor peso entre todas las palabras código distintas de cero.*

Lema 4.1.1. *[23] Sea $\mathscr{C}(n, k, d)$ un código lineal, entonces la distancia mínima es igual al peso mínimo de una palabra código distinta de cero.*

Demostración. Por definición tenemos que $wt(x) = d(0, x)$ y como el código es lineal, tenemos que $0 \in \mathscr{C}$. Sea c la palabra código con peso mínimo (que denotamos wt_{min}), tenemos entonces que $wt(c) = d(0, c) \geq d$. Por lo tanto $wt_{min} \geq d$.
Ahora demostremos que $wt_{min} \leq d$. Sean $c_1, c_2 \in \mathscr{C}$ tales que $d(c_1, c_2) = d$, como $\mathscr{C}$ es un código lineal, $(c_1 - c_2) \in \mathscr{C}$ y por lo tanto $d(c_1, c_2) = wt(c_1 - c_2) \geq wt_{min}$ y por lo tanto $wt_{min} \leq d$.

□

El problema que se presenta ahora es poder encontrar una función que codifique de k vectores a palabras código para enviarse por el canal. En la codificación sistemática, consiste en dejar que las primeras k coordenadas sean iguales a los símbolos del mensaje y las $n - k$ restantes las definimos de tal forma que haya una redundancia. Como sabemos todo subespacio puede ser representado explícitamente a partir de una base o con sus respectivas ecuaciones explicitas. Por lo que, ocurre algo similar en los códigos lineales, en donde se les puede representar a partir de su matriz generadora y de paridad.

Definición 4.1.8 (Matriz generadora). *[23] Una matriz generadora G para un código lineal $\mathscr{C}(n, k)$, es una matriz de tamaño $k \times n$ cuyas filas forman una base de $\mathscr{C}$.*

La matriz G no es única. Como G tiene rango k, esto significa que tiene k columnas linealmente independientes. Mediante operaciones por filas válidas podemos reescribir la matriz G, lo cual no altera al código sino representa otra asignación. Realizando estas operaciones podemos obtener de estas k columnas la matriz identidad. Además si estas k columnas coinciden con las k primeras columnas, entonces decimos que la matriz G se encuentra en la forma estándar, es decir $G = (I|A)$.

Definición 4.1.9 (Matriz de paridad). *[46] La matriz de verificación de paridad H para un código lineal $\mathscr{C}(n,k)$, es una matriz de tamaño $(n-k) \times n$ donde sus filas son linealmente independientes.*

La siguiente propiedad la cumplen las matrices de paridad: Si G es la matriz generadora y H la matriz de paridad, respectivamente de un código lineal $\mathscr{C}(n,k)$, entonces $GH^T = 0$.

Donde 0 es una matriz de ceros de tamaño $k \times (n-k)$.

La forma como se puede encontrar la matriz H es de la siguiente forma. Si $G = (I|A)$ es una matriz generadora en la forma estándar de un código lineal $\mathscr{C}(n,k)$, entonces se puede construir la matriz H de $\mathscr{C}$ como $H = (-A^T|I)$.

Una explicación de por qué se escoge la matriz de paridad de esta forma es la siguiente. Supongamos $u = (u_1, u_2, \ldots, u_k)$ el mensaje de tamaño de k que se quiere enviar por el canal. Lo que debemos hacer es convertir el mensaje u en una palabra código y para ello lo que realizamos es la multiplicación del mensaje con la matriz generadora del código. Así, obtenemos la siguiente palabra código:

$$c = (c_1, c_2, \ldots, c_k, x_1, x_2, \ldots, x_{n-k}).$$

Los x_i, para $1 \leq i \leq n-k$, generalmente son referidos como los símbolos de chequeo ya que proveen la redundancia necesaria para poder detectar y corregir los posibles errores. Sabiendo que Hc^T debe ser el vector de ceros si c es una palabra código, dados $u_i, 1 \leq i \leq k$ y $x_i, 1 \leq i \leq n-k$, puede determinarse de manera única

a partir de las ecuaciones del sistema $Hc^T = 0$.

Por lo tanto, los símbolos de chequeo x_i son determinados si tenemos una matriz de paridad H para el código $\mathscr{C}$.

Definición 4.1.10. *[46] El código dual $\mathscr{C}^\perp$ de un código lineal $\mathscr{C}(n,k)$ es definido como*

$$\mathscr{C}^\perp = \{x \in \mathbb{F}^n | x \times c = 0 \forall c \in \mathscr{C}\}.$$

Es claro que el código $\mathscr{C}^\perp$ es un subespacio lineal de dimensiones $n-k$. Luego el código $\mathscr{C}^\perp(n, n-k, d^\perp)$, donde $d^\perp$ denota la distancia mínima de $\mathscr{C}^\perp$ es un código lineal. Debido a la condición $GH^T = 0$, podemos ver fácilmente que la matriz generadora para el código dual es la matriz de paridad H del código $\mathscr{C}$ y la matriz de paridad es la matriz generadora G de $\mathscr{C}$.

4.2. Decodificación

Se quiere recuperar el mensaje u de y, el cual fue enviado por un canal como se puede ver la figura 4.1. Sin embargo, el mensaje recuperado puede ser diferente y esto se puede deber al hecho que el receptor no recupera $\mathscr{C}$ de y sino otra palabra código.

Intentaremos recuperar la palabra código más *cercana* a y, ya que de esta forma podemos afirmar cual es la palabra código $\mathscr{C}$ enviado. Para poder entender la noción de *cercano* utilizaremos la Definición 4.1.4.

En este punto nos podríamos preguntar sobre la capacidad que tiene nuestro código de poder detectar y corregir errores. Para ello vamos a introducir los siguientes lemas para poder luego dar una propiedad sobre esta cota de corrección.

Lema 4.2.1. *[23] Sea $\mathscr{C}(n,k,d)$ un código lineal con matriz de paridad H. Si hay j columnas linealmente dependientes, $\mathscr{C}$ contiene una palabra código con elementos distintos*

de ceros en algunas de las posiciones correspondientes.

Demostración. Sea $\mathscr{C}$ un código lineal con matriz de paridad H y $c = (c_1, c_2, \ldots, c_n)$. Por definición se tiene que $H \cdot c^T = 0$. De donde

$$\begin{aligned} h_{11}c_1 + \ldots + h_{1n}c_n &= 0 \\ &\vdots \\ h_{(n-k)1}c_1 + \ldots + h_{(n-k)n}c_n &= 0 \end{aligned}$$

si denotamos $c_{i_1}, \ldots, c_{i_r}$ con $i_r \leq n$, las entradas no nulas de c. Luego la anterior matriz es equivalente a

$$\begin{aligned} h_{1i_1}c_{i_1} + \ldots + h_{1i_r}c_{i_r} &= 0 \\ &\vdots \\ h_{(n-k)i_1}c_{i_1} + \ldots + h_{(n-k)i_r}c_{i_n} &= 0 \end{aligned}$$

Por lo tanto las columnas de H corresponden a las coordenadas no nulas de c que son linealmente dependientes. □

Lema 4.2.2. *[46] Sea $\mathscr{C}(n, k, d)$ un código lineal con una matriz de paridad H. La distancia mínima d es igual al menor número de columnas dependientes de H, es decir, que para cada selección de $d-1$ columnas es linealmente independientes y existen al menos una selección de d columnas linealmente dependientes.*

Demostración. Sea $H_0, \ldots, H_{n-1}$ columnas de H. Para demostrar el lema debemos considerar los siguientes dos casos:

1. Vamos a demostrar que no existe una selección de $d-1$ columnas linealmente dependientes en H. Para ello, asumamos lo contrario: existe una selección $H_{i_1}, \ldots, H_{i_{d-1}}$ de $d-1$ columnas linealmente dependientes. Por consiguiente,

existe una palabra $y = (y_0, \ldots, y_{n-1})$ con $wt(y) \leq d - 1$ y

$$0 = \sum_{j=1}^{d-1} y_{i_j} H_{i_j} = \sum_{v=0}^{n-1} y_v H_v = yH^T$$

Esto implica que y es una palabra código, lo cual contradice al Lema 4.1.1. Por lo tanto nuestra hipótesis es incorrecta.

2. Ahora vamos a demostrar que existe una selección de d columnas dependientes en H. Existe una palabra código $a = (a_0, \ldots, a_{n-1})$ con $wt(a) = d$. Ya que

$$0 = aH^T = \sum_{v=0}^{n-1} a_v h_v$$

Los d índices v con $a_v \neq 0$ determinan una selección de d columnas linealmente dependientes en H.

□

Teorema 4.2.1 (The singleton bound)**.** *[46] Sea $\mathscr{C}$ un código lineal (n, k) con distancia mínima d, entonces $d \leq n - k + 1$.*

Demostración. Cada $(n - k) \times (n - k + 1)$ submatriz de una matriz de paridad H tiene rango menor o igual que $n - k$, entonces cada conjunto de $n - k + 1$ columnas de una matriz de paridad son linealmente dependientes. Utilizando el lema anterior se concluye que $d \leq n - k + 1$ □

Definición 4.2.1 (Algoritmo de codificación). *[46] Dado un código $\mathscr{C}$, un vector $y \in \mathbb{F}^n$ y un entero positivo w, un algoritmo de codificación para $\mathscr{C}$, es un algoritmo que da el conjunto de todos los elementos $x \in \mathscr{C}$ de manera que $dist(x, y) \leq w$. El conjunto es vació si no existe c.*

1. **Algoritmos de codificación única:** Para tener una solución de única del problema de decodificación debemos analizar el número máximo de errores que el código lineal $\mathscr{C}$ con una distancia mínima puede corregir. Este número es

igual a $t = \lfloor \frac{d-1}{2} \rfloor$, esto se debe a que si consideramos c la palabra enviada y y la recibida, si se han cometido a lo sumo esta cantidad de errores entonces y pertenece a la bola centrada en c y de radio t y por tanto se corregirá con éxito.

2. **Lista de algoritmos de codificación:** Este caso se presenta cuando tenemos un entero $w > \lfloor \frac{d-1}{2} \rfloor$ y una palabra recibida y. La idea es dar una lista de palabras código $c_i \in \mathscr{C}$ tal que la distancia entre este c y y sea mínima. Para esto surgen dos posibilidades, una es tener la lista vacía y otra es tener una lista que contenga mas de una palabra código.

3. **síndrome:** Otro de los algoritmos de decodificación eficiente pero costoso computacionalmente para los códigos lineales es el algoritmo de decodificación por síndrome. Este algoritmo es uno de los algoritmos más utilizados en la literatura.

   Definición 4.2.2. *[46] Sea H una matriz de paridad para el código lineal $\mathscr{C}(n,k)$. El síndrome de y es $syn(y) = Hy^t$*

   Cabe mencionar que el síndrome de una palabra código $c \in \mathscr{C}$ es cero por definición.

   Teorema 4.2.2. *Sea $\mathscr{C}(n,k)$ un código lineal, $y = c + e$ una palabra recibida, con $c \in \mathscr{C}$ y e el error generado. Entonces $syn(y) = syn(e)$.*

   Demostración. Tomando la Definición 4.2.2 se tiene que:

$$syn(y) = syn(c) + syn(e) = syn(e)$$

   De la observación mencionada anteriormente se obtiene que $syn(c) = 0$. □

4.3. Códigos lineales

4.3.1. Códigos cíclicos

Los códigos cíclicos se presentan como un caso particular de los códigos lineales para los que existen algoritmos eficientes tanto para la codificación como para la decodificación de mensajes. Los procesos que hemos mencionado anteriormente pueden llevar a pensar que el costo computacional de la detección y corrección de errores puede ser baja. Sin embargo, en el proceso de decodificación puede elevarse el costo, debido a los casos en donde se presenta valores grandes para n, k y $(n-k)$, ya que se quiere realizar búsquedas de palabras códigos o síndromes que estarán almacenadas en tablas de memoria de gran tamaño.

A continuación comenzaremos con el estudio de una clase importante conocida como *códigos cíclicos*. Muchas familias de códigos son considerados códigos cíclicos o una extensión.

Definición 4.3.1 (Códigos cíclicos). *[23] Un código lineal $\mathscr{C}$ es un código cíclico si para todo $u = (u_0, u_1, \ldots, u_{n-1}) \in \mathscr{C}$, entonces $(u_{n-1}, u_0, u_1, \ldots, u_{n-2}) \in \mathscr{C}$.*

En el estudio de los códigos cíclicos de tamaño n, es conveniente etiquetar las posiciones de la siguiente forma $0, 1, \ldots, n-1$ y pensar que nos encontramos en los enteros módulo n. Un código lineal $\mathscr{C}$ de tamaño n sobre $\mathbb{F}_q$ es cíclico siempre que para cada vector $c = (c_0, \ldots c_{n-2}, c_{n-1}) \in \mathscr{C}$ el vector $c' = (c_{n-1}, c_0, \ldots, c_{n-2})$ obtenido de c por el desplazamiento $i \to i+1 \mod n$ de sus coordenadas, se tiene que $c' \in \mathscr{C}$. Lo anterior implica que un código cíclico contiene todos los n cambios cíclicos de una palabra código. Por lo tanto, es conveniente pensar cíclicamente en las posiciones de las coordenadas, donde una vez que se llega a la coordenada $n-1$ se vuelve a comenzar con la coordenada 0.

Cuando se investiga los códigos cíclicos sobre $\mathbb{F}$, en la mayoría de los casos se representan las palabras código en forma polinomial. Esto se debe a que existe una

correspondencia biyectiva entre el vector $c = (c_0, c_1, \ldots, c_{n-1})$ en $\mathbb{F}_q^n$ y el polinomio definido de la siguiente forma $c(x) = c_0 + c_1 x + \ldots + c_{n-1}x^{n-1} \in \mathbb{F}[q]$ de grado a lo sumo $n-1$.

Note que si $c(x) = c_0 + c_1 x + \ldots + c_{n-1}x^{n-1}$, entonces:

$$xc(x) = c_{n-1}x^n + c_0 x + c_1 x^2 + \ldots + c_{n-2}x^{n-1},$$

lo cual representaría la palabra código desplazada un espacio hacia la derecha si x^n sea igual a 1. Más formalmente, el hecho que un código cíclico C sea invariante bajo un desplazamiento cíclico, implica que si $c(x)$ está en $\mathscr{C}$, entonces también está $xc(x)$ siempre que multiplicamos modulo $x^n - 1$. Lo cual sugiere estudiar los códigos cíclicos en un anillo cociente, de la forma $R_n = \mathbb{F}_q[x]/(x^n - 1)$.

En la siguiente propiedad no distinguimos entre una palabra código o en su representación por medio de un polinomio.

Lema 4.3.1. *[47] Si* $c(x) = c_{n-1}x^{n-1} + \ldots + c_1 x + c_0$ *y* $c'(x) = c_{n-2}x^{n-1} + \ldots + c_0 x + c_{n-1}$. *Entonces*

$$c'(x) = xc(x) - c_{n-1}(x^n - 1).$$

Demostración. Para demostrar esta propiedad basta con realizar la operación. Entonces:

$$\begin{aligned} c'(x) &= xc(x) - c_{n-1}(x^n - 1) \\ &= c_{n-1}x^n + c_{n-2}x^{n-1} + \ldots + c_1 x^2 + c_0 x - c_{n-1}x^n + c_{n-1} \\ &= c_{n-2}x^{n-1} + \ldots + c_1 x^2 + c_0 x + c_{n-1}. \end{aligned}$$

□

Teorema 4.3.1. *[20] Sea* $\mathscr{C}$ *un código cíclico* $(n, k) \in \mathbb{F}^n$ *y* $g(x)$ *un polinomio mónico de grado mínimo en* $\mathscr{C} \setminus \{0\}$, *entonces:*

1. $g(x)$ *divide a* $c(x)$ *para cada* $c(x) \in \mathscr{C}$.

2. $g(x)$ *divide a* $x^n - 1$ *en* $\mathbb{F}_q[x]$.

3. $k = n - deg(g(x))$.

Demostración.

1. Supongamos $c(x) \in \mathscr{C}$, entonces $c(x) = a(x)g(x) + r(x)$ donde se tiene que $deg(r(x)) < deg(g(x))$. Ya que $deg(a(x)) \leq n - 1 - s$ tenemos que es una palabra código y por lo tanto $r(x) = c(x) - a(x)g(x)$ esta en $\mathscr{C}$. Por tanto $r(x) = 0$, esto se debe a que $r(x)$ tiene grado n y esto es posible solamente en el caso cuando $r(x) = 0$ y en consecuencia $g(x)$ divide a $c(x)$.

2. En $\mathbb{F}_q[x]$, tenemos entonces que $x^n - 1 = a(x)g(x) + r(x)$ donde se tiene que $deg(r(x)) < deg(g(x))$. Ahora, $r(x) \equiv -a(x)g(x) \bmod(x^n - 1)$, así $r(x) \in \mathscr{C}$ lo cual es imposible al menos que $r(x) = 0$. En consecuencia $g(x)$ divide a $x^n - 1$.

3. Tenemos que por definición de $g(x)$ que $k \leq n - deg(g(x))$ ya que solo hay $n - deg(g(x))$ posiciones a la izquierda. Si $g(x)$ tiene grado s, entonces

$$g(x) = g_{s-1}x^{s-1} + \ldots + g_1x + g_0 + x^s$$

   del Lema 4.3.1, tenemos que $x^j g(x)$ esta en $\mathscr{C}$ si $j \leq n - 1 - s$, por lo tanto $a(x)g(x)$ donde $deg(a(x)) \leq n - 1 - s$ esta en $\mathscr{C}$.
   Por otro lado si $x^j g(x)$ son palabras códigos linealmente independientes de $\mathscr{C}$, entonces $k \geq n - s$ y por tanto $k = n - s$.

□

Note que $g(x)$ es el llamado polinomio generador del código cíclico $\mathscr{C}$. Además $g(x)$ es único ya que si existiera otro, la diferencia entre ellos tendría un grado más abajo, debido a que $\mathscr{C}$ es lineal y sería una palabra código. Del anterior teorema podemos ver que un código cíclico corresponde a un divisor de $x^n - 1$ y nos podría-

mos preguntar si cualquier divisor de $x^n - 1$ también le corresponde algún código cíclico.

Teorema 4.3.2. *[20] Supongamos que $g(x) \in \mathbb{F}[x]$ es mónico y divide a $x^n - 1$ entonces*

$$\mathscr{C} = \{i(x)g(x) | i(x) \in \mathbb{F}[x], deg(i(x)) < n - deg(g(x))\}$$

Es un código cíclico con polinomio generador $g(x)$.

Demostración. Claramente $\mathscr{C}$ es un código lineal, si $\mathscr{C}$ es cíclico tiene un polinomio generador $g(x)$ y ya que la dimensión es $n - deg(g(x))$, entonces solo queda demostrar que $\mathscr{C}$ es cíclico. Para ello

$$g(x) = g_{s-1}x^{s-1} + \ldots + g_1 x + g_0 + x^s$$

$$h(x) = \frac{x^n - 1}{g(x)} = x^{n-s} + h_{n-s-1}x^{n-s-1} + \ldots + h_1 x + h_0.$$

Sea $c(x) = i(x)g(x)$ donde $deg(i(x)) < n - s$, entonces

$$c' = xc(x) - c_{n-1}(x^n - 1) = xi(x)g(x) - c_{n-1}h(x)g(x)$$

por lo que

$$c'(x) = (xi(x) - x_{n-1}h(x)g(x).$$

Ahora $c_{n-1} = i_{n-s-1}$ ya que queremos que el grado se mantenga lo que implica que $c'(x) \in \mathscr{C}$. □

En $\mathbb{F}_q[x]/(x^n - 1)$ hay mas polinomios que cumplen la propiedad anterior, un ejemplo de ello es considerar la factorización de $x^7 - 1$ en $\mathbb{F}_2[x]$, luego los factores irreducibles están dados por

$$x^7 - 1 = (x + 1)(x^3 + x + 1)(x^3 + x^2 + 1).$$

Ya que $x^7 - 1$ tiene tres factores irreducibles sobre $\mathbb{F}_2$, entonces se tienen 2^3 códigos cíclicos distintos de tamaño 7. Algunos pueden ser equivalentes.

Teorema 4.3.3. *Sea $\mathscr{C}(n,k)$ un código cíclico en $\mathbb{F}_q$ con polinomio generador $g(x)$, entonces el grado de $g(x)$ es igual a $n-k$, escribimos $g(x) = g_0 + g_1x + \ldots + g_{n-k}x^{n-k}$. Entonces una matriz generadora G para $\mathscr{C}$ esta dado por*

$$G = \begin{pmatrix} g_0 & g_1 & \cdots & g_{n-k} & 0 & 0 & \cdots & 0 \\ 0 & g_0 & g_1 & \cdots & \cdots & g_{n-k} & 0 & \cdots & 0 \\ 0 & 0 & g_0 & g_1 & \cdots & \cdots & g_{n-k} & \cdots & 0 \\ \vdots & \vdots & \cdots & \ddots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 & g_0 & g_1 & \cdots & \cdots & g_{n-k} \end{pmatrix}.$$

Demostración. Supongamos que el grado de $g(x) = l$. Entonces consideremos la matriz G^* que tiene por dimensión $(n-l) \times l$ donde cada fila representa una palabra código $x^i g(x)$, para $0 \leq i \leq n-l$, además como $g(x)$ es un polinomio mononico entonces el rango de G^* es igual a $n-l$.

Ya que cada palabra código debe ser divisible por $g(x)$, esto puede ser escrito como $u(x)g(x)$. Pero cada palabra código tiene grado menor que n, por lo que las palabras código las podemos escribir como $u(x)g(x)$ tal que $deg(u(x))$ es menor que $n-l$. En otras palabras, cada palabra código es una combinación lineal de las filas de G^*. Ya que $\mathscr{C}$ tiene dimensión k, podemos concluir que $k = n-l$ y que $G^* = G$. □

4.3.2. Códigos Reed Solomon

En enero de 1959, Irving Reed y Gaus Solomon, enviaron un documento al *Jorunal of the society for industrial and applied mathematics*[35]. En el documento se hablaba de una nueva clase de códigos correctores de errores que ahora son llamados los códigos Reed Solomon. Estos códigos tienen una gran importancia en el desarrollo de las telecomunicaciones en el siglo veinte.

Definición 4.3.2. *[23] Sean $x_1, x_2, \ldots, x_n$ elementos diferentes de un campo finito $\mathbb{F}_q$. Para $k \leq n$ considere el conjunto P_k de polinomios en $\mathbb{F}[x]$ de grado menor a k. En el Reed Solomon las palabras códigos se forman de la siguiente forma:*

$$(f(x_1), f(x_2), \ldots, f(x_n)) \text{ donde } f \in P_k.$$

A estos códigos los denotaremos como los $RS(n, k)$.

Un conjunto de palabras código es construido dejando que los k símbolos de información tomen todos los valores posibles, ya que los símbolos son tomados de $\mathbb{F}_q$, entonces podemos tomar q valores diferentes por lo que tenemos que hay q^k palabras códigos en código Reed Solomon.

Como vimos al principio de este capítulo un código se dice que es lineal si la suma de cualesquiera dos palabras códigos es también una palabra código. Además para un elemento en $\mathbb{F}_q$ la multiplicación con alguna palabra código es también una palabra código. Lo anterior lo podemos ver como:

$$c_1 = (f_1(x_1), f_1(x_2), \ldots, f_1(x_n))$$
$$c_2 = (f_2(x_1), f_2(x_2), \ldots, f_2(x_n)).$$

Entonces

$$ac_1 + bc_2 = (g(x_1), g(x_2), \ldots, g(x_n)).$$

Donde $a, b \in F_q$ y $g(x) = af_1(x) + bf_2(x)$.

Ahora bien, el peso de una palabra código es al menos $n - k + 1$ y esto se debe a que un polinomio de grado menor a k tiene a lo sumo $k - 1$ ceros. Tomando este hecho y la cota singleton, introducimos el siguiente teorema.

Teorema 4.3.4. *[23] La distancia mínima de un código Reed Solomon(n,k) es $n-k+1$.*

Demostración. Cada polinomio de grado $k-1$ sobre $\mathbb{F}_q[x]$ tiene a lo sumo $k-1$ (no necesariamente distintas) raíces y si dos polinomios coinciden en más que $k-1$ lugares, entonces deben ser el mismo polinomio. □

Definiremos la matriz generadora para un Reed Solomon, la cual existe ya que por definición sabemos que estos códigos son lineales. La estructura algebraica de los códigos Reed Solomon significa que tanto para codificar y decodificar puede ser extremadamente eficiente. Porque los elementos diferentes de cero de un campo finito puede ser escritos como $\alpha, \alpha^2, \ldots, \alpha^{q-1}$, la matriz generadora de el código tiene una estructura particular, la cual es:

$$\begin{pmatrix} 1 & 1 & \cdots & 1 \\ \alpha_1 & \alpha_2 & \cdots & \alpha_n \\ \vdots & \vdots & \ddots & \vdots \\ \alpha_1^{k-1} & \alpha_2^{k-1} & \cdots & \alpha_n^{k-1} \end{pmatrix} \tag{1}$$

Las columnas de la matriz generadora corresponden a los monomios $1, X, X^2, X^3, \ldots, X^{k-1}$. Observe que si

$$f(X) = f_0 + f_1 X + \ldots + f_{k-1} X^{k-1}$$

entonces

$$G \times \begin{bmatrix} f_0 \\ f_0 \\ \vdots \\ f_{k-1} \end{bmatrix} = \begin{bmatrix} f_0 \\ f_\alpha \\ \vdots \\ f_{\alpha^{k-1}} \end{bmatrix}. \tag{2}$$

La estructura de G permite codificar extremadamente rápido, ya que existe algorit-

mos que calculan $G \times f$ en tiempo $O(q \cdot \log(k))$.

4.3.3. Códigos Reed Solomon Generalizados

Los códigos Reed Solomon Generalizados son muy usados en la practica y además las matemáticas que fundamentan su construcción son importantes para nuestro estudio en los siguientes capítulos. En la anterior sección mencionamos algunas definiciones y propiedades básicas de los Reed Solomon. Ahora introduciremos los códigos Reed Solomon generalizados.

Definición 4.3.3. *[27] Sea* $\mathbb{F}$ *un campo y* $\alpha = (\alpha_1, \alpha_2, \ldots, \alpha_n)$ *tal que* $\alpha_i \in (\mathbb{F}_q)^n$ *y además todos los* x_i *son diferentes entre ellos. Sea* $v = (v1, v_2, \ldots, v_n)$*, donde* v_i *son elementos diferentes de cero de* $(\mathbb{F}_q^*)^n$*. Para* $0 \leq k \leq n$*, definimos los códigos Reed Solomon generalizados como*

$$GRS(\alpha, v) = \{(v_1 f(\alpha_1), v_2 f(\alpha_2), \ldots, v_n f(\alpha_n) : f(x) \in \mathbb{F}_q[x] \; y \; \deg(f(x)) < k\}.$$

Los elementos $\alpha_1, \alpha_2, \ldots, \alpha_n$ son llamados los localizadores de código del $GRS(\alpha, v)$

Teorema 4.3.5. *[27] Sea* $GRS_k(\alpha, v)$ *un código lineal sobre* $\mathbb{F}_q$*, tenemos que dim* $= n - k + 1$.

Demostración. Es claro que el tamaño del código $GRS_k(\alpha, v)$ es n. Ahora vamos a demostrar que la mínima distancia es $n - k + 1$. Para ello consideremos a $f(x)$ diferente a la función constante cero. Ya que el grado de $\deg(f(x)) < k$, el polinomio $f(x)$ tiene como mucho $k - 1$ ceros, es decir, la palabra código $(v_1 f(\alpha_1), v_2 f(\alpha_2), \ldots, v_n f(\alpha_n))$ tiene a lo sumo una cantidad de $k - 1$ ceros en sus coordenadas. En otras palabras, su peso es como mínimo $n - (k - 1) = n - k + 1$, así la distancia mínima d del código $GRS(\alpha, v)$ satisface $d \geq n - k + 1$. Sin embargo, la cota Singleton muestra que $d \leq n - k + 1$, por lo que $d = n - k + 1$. □

En el caso cuando $v = (1, 1, \ldots, 1)$ y $n < q - 1$, el código Reed Solomon Generali-

zado se convierte en el código Reed Solomon el cual se menciono en la subsección anterior.

Ahora buscaremos una representación para nuestra matriz generadora. El argumento del teorema anterior, hace claro que cualquier base $f_1(\alpha), f_2(\alpha), \ldots, f_k(\alpha)$ de $\mathbb{F}_k[x]$ da lugar a una base $f_1, f_2, \ldots, f_k$ del código. En particular una buena base polinomial es el conjunto de monimios $1, x, \ldots, x^{k-1}$. La matriz generadora correspondiente a las i filas son $v_1\alpha_1^i, v_2\alpha_2^i, \ldots, v_n\alpha_n^i$, por lo que la matriz generadora es

$$\begin{pmatrix} v_1 & v_2 & \cdots & v_n \\ v_1\alpha_1 & v_2\alpha_2 & \cdots & v_n\alpha_n \\ \vdots & \vdots & \ddots & \vdots \\ v_1\alpha_1^{n-k-1} & v_2\alpha_2^{n-k-1} & \cdots & v_n\alpha_n^{n-k-1} \end{pmatrix}. \tag{3}$$

4.3.4. Códigos Alternantes

Hasta el momento hemos hablado de los códigos Reed Solomon y Reed Solomon generalizados sobre un campo. En esta subsección introduciremos los códigos alternantes, que es una familia de códigos la cual consiste en restringir los Reed Solomon generalizados a un subcampo. Antes de comenzar debemos introducir la definición de un subcódigo de subcampo, ya que con esto podremos definir los códigos alternantes.

Definición 4.3.4 (Subcódigo de subcampo). *[46] Si $\mathscr{C}$ es un código sobre $\mathbb{F}$ y $\mathbb{F}_{SUB}$ es un subcampo de $\mathbb{F}$, entonces el $\mathbb{F}_{SUB}$ es el código que consiste de todas las palabras de c, las cuales tienen entradas en $\mathbb{F}_{SUB}$.*

Definición 4.3.5 (Códigos alternantes). *[27] Un código alternante $A_k(\alpha, v')$ sobre el campo finito $\mathbb{F}_q$ es el subcódigo de subcampo $GRS_k(\alpha, v)|\mathbb{F}_q$, donde $GRS_k(\alpha, v)$ es un código Reed Solomon generalizado sobre $\mathbb{F}_{q^m}$, para $m \geq 1$.*

Proposición 4.3.1. *[27] Los códigos alternantes $A_k(\alpha, v')$ tienen como parámetro $[n, k', d]$ donde $mk - (m-1)n \leq k' \leq k$ y $d \geq n - k + 1$.*

Demostración. Por el Teorema (4.3.5), $GRS_k(\alpha, v)$ se tiene como parámetro

$$[n, k, n - k + 1].$$

Ya que $A_k(\alpha, v')$ claramente tiene tamaño n y su dimensión es k' entonces se tiene que $k \leq k'$. La parte siguiente del teorema se sigue por el hecho de ser un subcódigo de un subcampo. □

Ahora bien, tomando la definición anterior y la propiedad de la matriz de paridad del un código Reed Solomon generalizado, obtenemos que $A_k(\alpha, v')$ no es mas que:

$$\{\boldsymbol{c} \in \mathbb{F}_q^n : \boldsymbol{c}H^T = \mathbf{0}\}.$$

Donde H es la matriz de paridad del código $GRS_k(\alpha, v')$. Observe que H esta determinado por α y v', lo cual nos permitirá expresar el código alternante en términos de estas variables.

Recordemos ahora que los elementos $\beta \in \mathbb{F}_{q^m}$, los podemos escribir de la siguiente forma $\sum_{i=0}^{m-1} \beta_i \alpha^i$, donde α es un elemento primitivo de $\mathbb{F}_{q^m}$ y $\beta_i \in \mathbb{F}_q$ para todo i. Entonces, podemos ver que al momento de reemplazar cada entrada de la matriz H por el vector columna $(\beta_1, \beta_2, \cdots, \beta_n)^T$, podemos obtener una matriz $\overline{H}$ cuya dimensión es igual a $(n-k)m \times n$, donde cada entrada pertenece a $\mathbb{F}_q$ y se cumple que

$$\{\boldsymbol{c} \in \mathbb{F}_q^n : \boldsymbol{c}\overline{H^T} = \mathbf{0}\}.$$

Esta matriz $\overline{H}$ juega el rol de ser la matriz de paridad del código $A_k(\alpha, v')$, exceptuando que las filas no son necesariamente linealmente independiente, sin embargo, en esta tesis nos abstendremos de llamarla de esta forma ya que no cumple con todas las propiedades de una matriz de paridad.

Ejemplo 4.3.1. *Sean $q = 2, m = 3$ y $n = 6$. Definamos a α como el elemento primitivo de $\mathbb{F}_8$ que satisface $\alpha^3 + \alpha + 1 = 0$. Tomando a $\boldsymbol{v'} = (1, \cdots, 1)$ y $\alpha = (\alpha, \alpha^2, \cdots, \alpha^6)$. Entonces $A_3(\alpha, v') = \{\boldsymbol{c} \in \mathbb{F}_2^6 : \boldsymbol{c}\overline{H}^T = 0\}$, donde*

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ \alpha & \alpha^2 & \alpha^3 & \alpha^4 & \alpha^5 & \alpha^6 \\ \alpha^2 & \alpha^4 & \alpha^6 & \alpha & \alpha^3 & \alpha^5 \end{pmatrix}. \tag{4}$$

Entonces, tomando los posibles vectores en donde cada entrada $\beta \in \mathbb{F}_2$, conseguimos la matriz $\overline{H}$:

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 1 \end{pmatrix}. \tag{5}$$

Ahora si realizamos la correspondiente reducción obtenemos la siguiente matriz:

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}. \tag{6}$$

Ya que sabemos que los códigos alternantes son lineales, entonces podemos encontrar la matriz generadora de este código, esta matriz tiene la siguiente forma:

$$\begin{pmatrix} 1 & 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 \end{pmatrix}. \tag{7}$$

Por lo que el código alternante $A_k(\alpha, v')$ tiene como parámetros $(6, 2, 4)$.

4.3.5. Códigos de Goppa

Una de las subclases más interesantes de los códigos alternantes es la familia de los códigos de Goppa, introducida por V. D. Goppa [19] a principios de la década de 1970. Los códigos Goppa también se utilizan en criptografía, como veremos en el siguiente capítulo en los criptosistemas de McEliece y el criptosistema Niederreiter los cuales son ejemplos de criptosistemas a clave pública que utilizan estos códigos.

Definición 4.3.6. *[27] Sea $g(x)$ un polinomio en $\mathbb{F}_{q^m}[x]$ para algún m fijo, y sea $L = \{\alpha_1, \alpha_2, \ldots, \alpha_n\}$ un subconjunto de $\mathbb{F}_q$ tal que $L \cap \{\text{ceros de } g(x)\} = \emptyset$. Para $\boldsymbol{c} = (c_1, \ldots, c_n) \in$*

$\mathbb{F}_q^n$, *sea*

$$R_c(x) = \sum_{i=0}^{n-1} \frac{c_i}{x - \alpha_i}.$$

Los códigos de goppa $\Gamma(L, g)$ *son definidos como*

$$\Gamma(L, g) = \{\boldsymbol{c} \in \mathbb{F}_q^n : R_c(x) \equiv 0 \mod g(x)\}.$$

Llamaremos al polinomio $g(x)$ como el polinomio de Goppa. Cuando $g(x)$ es irreducible entonces $\Gamma(L, g)$ es llamado el código de Goppa irreducible.

Observe que el termino $\frac{1}{x-\alpha_i}$ se puede ver como un polinomio $p_i(x)$ modulo $g(x)$

$$\frac{1}{x - \alpha_i} \equiv p_i(x) = p_{i1} + p_{i2}x + \ldots + p_{it}x^{t-1} \mod g(x).$$

Esto se debe a la forma que tiene los elementos en el campo $\mathbb{F}_{q^m}/g(x)$. Por lo tanto podemos reescribir la ecuación de la Definición (4.3.6) como

$$\sum_{i=0}^{n-1} c_i p_i(x) \equiv 0 \mod g(x). \tag{8}$$

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

Teorema 4.3.6. *[27] Sea un código de Goppa* $\Gamma(L, g) = \{\boldsymbol{c} \in \mathbb{F}_q^n : cH'^T = \mathbf{0}\}$, *donde* $g(x)$ *tiene grado* t, $L = \{\alpha_1, \alpha_2, \ldots, \alpha_n\}$ *y definimos* H' *como*

$$H' = \begin{pmatrix} \alpha_1^{t-1}g(\alpha_1)^{-1} & \alpha_2^{t-1}g(\alpha_2)^{-1} & \cdots & \alpha_n^{t-1}g(\alpha_n)^{-1} \\ \alpha_1^{t-2}g(\alpha_1)^{-1} & \alpha_2^{t-2}g(\alpha_2)^{-1} & \cdots & \alpha_n^{t-2}g(\alpha_n)^{-1} \\ \vdots & \vdots & \ddots & \vdots \\ \alpha_1 g(\alpha_1)^{-1} & \alpha_2 g(\alpha_2)^{-1} & \cdots & \alpha_n g(\alpha_n)^{-1} \\ g(\alpha_1)^{-1} & g(\alpha_2)^{-1} & \cdots & g(\alpha_n)^{-1} \end{pmatrix} \tag{9}$$

entonces H' es una matriz de paridad.

Demostración. Para demostrar que H' es una matriz de paridad del código de Goppa, debemos recordar que si $\boldsymbol{c} \in \Gamma(L, g)$ si y solo si se cumple (8) donde podemos escribir $\frac{1}{x-\alpha_i} \equiv p_{i1} + p_{i2} + \ldots + p_{it}x^{t-1} \mod g(x)$. Una de las posibles matrices de paridad debe cumplir que $cH^T = 0$, por lo que proponemos

$$H = \begin{pmatrix} p_{11} & \cdots & p_{n1} \\ \vdots & \ddots & \vdots \\ p_{1t} & \cdots & p_{nt} \end{pmatrix}. \tag{10}$$

Verificando que $cH^T = 0$, tenemos

$$(c_1, \ldots, c_n) \times \begin{pmatrix} p_{11} & \cdots & p_{n1} \\ \vdots & \ddots & \vdots \\ p_{1t} & \cdots & p_{nt} \end{pmatrix} = (\sum_i^n c_i p_{i1}, \ldots, \sum_i^n c_i p_{it}). \tag{11}$$

Podemos ver que la ecuación (11) es cero modulo $g(x)$, por definición. Luego para poder determinar los factores p_{ij} de la matriz H debemos reescribir el termino $\frac{1}{x-\alpha_i}$ de la siguiente manera, primero definamos $u(x)$ de la siguiente manera

$$u(x) = -g(x)g(\alpha_i)^{-1} + 1 = -(g(x) - g(\alpha_i))g(\alpha_i)^{-1}$$

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

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

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

$$p_i = -\frac{gt \times (x^t - \alpha_i^t) + \ldots + g_1 \times (x - \alpha)}{x - \alpha_i} \times h_i. \tag{12}$$

La fracción en (12) puede ser escrita como

$$g_t(x^{t-1} + x^{t-2}\alpha_i + \ldots + \alpha_i^{t-1}) + g_{t-1}(x^{t-2} + x^{t-3}\alpha_i + \ldots + \alpha_i^{t-2}) + \ldots + g_2(x - \alpha_i) + g_1.$$

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

$$\begin{cases} p_{i1} & = & -(g_t\alpha_i^{t-1} + g_{t-1}\alpha_i^{t-2} + \ldots + g_2\alpha_i + g_1)h_i \\ p_{i2} & = & -(g_t\alpha_i^{t-2} + g_{t-1}\alpha_i^{t-3} + \ldots + g_2)h_i \\ & \vdots & \\ p_{i(t-1)} & = & -(g_t\alpha_i + g_{t-1})h_i \\ p_{it} & = & -(g_t)h_i \end{cases} \tag{13}$$

Combinando 10 y 13, encontramos que $H = ABD$ para

$$A = \begin{pmatrix} -g_t & -g_{t-1} & -g_{t-1} & \cdots & -g_1 \\ 0 & -g_t & -g_{t-1} & \cdots & -g_2 \\ 0 & 0 & -g_t & \cdots & -g_3 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & -g_t \end{pmatrix} \tag{14}$$

$$B = \begin{pmatrix} \alpha_1^{t-1} & \alpha_2^{t-1} & \cdots & \alpha_n^{t-1} \\ \alpha_1^{t-2} & \alpha_2^{t-2} & \cdots & \alpha_n^{t-2} \\ \vdots & \vdots & \ddots & \vdots \\ \alpha_1 & \alpha_2 & \cdots & \alpha_n \\ 1 & 1 & \cdots & 1 \end{pmatrix} D = \begin{pmatrix} h_1 & 0 & 0 & \cdots & 0 \\ 0 & h_2 & 0 & \cdots & 0 \\ 0 & 0 & h_3 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & h_n \end{pmatrix} \tag{15}$$

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

Los parámetros de un código de Goppa son el tamaño n, la dimensión k y la distancia mínima d. Usaremos la notación (n, k, d) para un código Goppa con parámetros n, k y d. El primer parámetro, n es la longitud de las palabras de código c y por lo tanto está fijada por L. Para los otros dos parámetros, se pueden derivar límites inferiores.

Colorario 4.3.1. *[27] Para un polinomio de Goppa $g(x)$ de grado t y $L = (\alpha_1, \alpha_2, \ldots, \alpha_n\}$, el código de Goppa $\Gamma(L, g)$ es un código lineal sobre $\mathbb{F}_q$ con parámetros $[n, k, d]$, donde se tiene $k \geq n - mt$ y $d \geq t + 1$.*

Ejemplo 4.3.2. *Sea $q = 2$ y tome $g(x) = \alpha^3 + x + x^2$, donde α es un elemento primitivo de $\mathbb{F}_8$, que satisface $\alpha^3 + \alpha + 1 = 0$. Sea $L = \mathbb{F}_8$, así $n = 8$ y $m = 3$. Entonces definimos el código $\Gamma(L, g) = \{\boldsymbol{c} \in \mathbb{F}_2^8 : \boldsymbol{c}H^T = \boldsymbol{0}$, donde*

$$\begin{pmatrix} \alpha^4 & \alpha^4 & \alpha & 1 & \alpha & \alpha^2 & \alpha^2 & 1 \\ 0 & \alpha^4 & \alpha^2 & \alpha^2 & \alpha^4 & \alpha^6 & 1 & \alpha^6 \end{pmatrix}. \tag{16}$$

De la misma forma al ejemplo del código alternante, debemos reemplazar cada entrada en H por el vector columna en $\mathbb{F}_2^3$, lo cual nos permite obtener una matriz $\overline{H}$ lo cual nos da la siguiente la matriz

$$\begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 1 \end{pmatrix}. \tag{17}$$

Aplicando la reducción por filas, obtenemos al siguiente matriz en su forma estándar:

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \end{pmatrix}. \tag{18}$$

Así la matriz generadora del código de Goppa con parámetros $(8, 2, 5)$ *es igual a*

$$\begin{pmatrix} 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 \end{pmatrix}. \tag{19}$$

4.4. Codificación de los códigos de Goppa

La codificación de los códigos de Goppa supone multiplicar el mensaje por la matriz generadora del código.

Definición 4.4.1. *La matriz generadora de un código de Goppa* $\Gamma(L, g)$ *se define como la matriz G de tamaño* $k \times n$ *tal que las filas de G forman una base para el código.*

Con el fin de enviar un mensaje usando los códigos de Goppa, el mensaje es primero escrito en bloques de k símbolos. Entonces, cada bloque es multiplicado por la

matriz generadora G. El vector resultante forman el conjunto de palabras códigos, es decir:

$$(m_1, m_2, \cdots, m_k) \times G = (c_1, c_2, \cdots, c_n).$$

4.5. Decodificación de los códigos de Goppa

A continuación vamos a discutir un algoritmo de decodificación de los códigos binarios de Goppa propuesto 1975 por Nicholsa J. Patterson [34]. Supongamos que recibimos un vector y que contiene errores. Como hemos visto antes, podemos escribir y como una palabra código c más un vector de error e: $y = c + e$

Recordemos la definición (4.3.6) del código Goppa $R_y(x) = \sum_{i=0}^{n-1} \frac{y_i}{x-\alpha_i}$. Dado que y puede contener errores, obtenemos la siguiente ecuación para $R_y(x)$

$$R_y(x) = \sum_{i=0}^{n-1} \frac{c_i + e_i}{x - \alpha_i} = \sum_{i=0}^{n-1} \frac{c_i}{x - \alpha_i} + \sum_{i=0}^{n-1} \frac{e_i}{x - \alpha_i}. \tag{20}$$

Por la definición (4.3.6) sabemos que $\sum_{i=0}^{n-1} \frac{c_i}{x-\alpha_i} \equiv 0 \mod g(x)$, luego

$$R_y(x) = \sum_{i=0}^{n-1} \frac{e_i}{x - \alpha_i} \mod g(x).$$

Es decir el síndrome de y. Para poder encontrar los errores que ocurrieron durante la transmisión, definimos un polinomio de localización de errores.

Definición 4.5.1. *Sea $\sigma(x)$ el polinomio localizador de errores definido de la siguiente manera:*

$$\sigma(x) = \prod_{i|e_i \neq 0} (x - \alpha_i).$$

Donde e_i significa el bit en el índice i del vector de error, por lo que el producto solo contiene el término $(x - \alpha_i)$ si $e_i = 1$. Por lo tanto si tenemos el polinomio $\sigma(x)$,

podemos evaluarlo en α_i y así saber si en la posición i hay un error o no. De esta manera podemos reconstruir el vector error e.

Ahora, para obtener este polinomio localizador de errores vamos a utilizar una representación distinta de $R_y(x)$. Empecemos por combinar la suma de fracciones en una sola. Por ejemplo, tomando la suma de algunas fracciones aleatorias $\frac{1}{2}$, $\frac{1}{3}$ y $\frac{1}{5}$, podemos combinar esto en una sola fracción.

$$\begin{aligned}\frac{1}{2}+\frac{1}{3}+\frac{1}{5} &= \frac{1\cdot 3\cdot 5}{2\cdot 3\cdot 5}+\frac{1\cdot 2\cdot 5}{2\cdot 3\cdot 5}+\frac{1\cdot 2\cdot 3}{2\cdot 3\cdot 5}\\ &= \frac{1\cdot 3\cdot 5+1\cdot 2\cdot 5+1\cdot 2\cdot 3}{2\cdot 3\cdot 5}\\ &= \frac{\sum_{j\in\{2,3,5\}} 1\cdot \prod_{i\neq j} i}{\prod_{i\in\{2,3,5\}} i}.\end{aligned}$$

Podemos ver que el nuevo numerador de $\frac{1}{2}$ es la multiplicación con todos los otros denominadores 3 y 5. De la misma manera, los numeradores de $\frac{1}{3}$ y $\frac{1}{5}$ se convierte en la multiplicación con los otros denominadores. Si los numeradores son variables binarias en lugar de simplemente 1, tendríamos lo siguiente:

$$\frac{c_2}{2}+\frac{c_3}{3}+\frac{c_5}{5} = \frac{\sum_{j\in\{2,3,5\}} c_j\cdot \prod_{i\neq j} i}{\prod_{i\in\{2,3,5\}} i}. \tag{21}$$

Veamos que pasa si uno de los valores es cero. Si tomamos $c_2 = 1$, $c_3 = 1$ y $c_5 = 0$, entonces

$$\frac{1}{2}+\frac{1}{3}+\frac{0}{5}=\frac{1\cdot 3\cdot 5}{2\cdot 3\cdot 5}+\frac{1\cdot 2\cdot 5}{2\cdot 3\cdot 5}+\frac{0\cdot 2\cdot 3}{2\cdot 3\cdot 5}$$
$$=\frac{1\cdot 3\cdot 5+1\cdot 2\cdot 5}{2\cdot 3\cdot 5}.$$

Vemos que el denominador que pertenece a c_5, que en este caso era 5, apareció en cada término del numerador y apareció en el denominador, por lo que se puede cancelar. Si queremos incorporar esta cancelación en la representación del producto en (21), podemos usar c_i como exponente.

$$\frac{\sum_{j\in\{2,3,5\}} c_j\cdot\prod_{i\neq j} i}{\prod_{i\in\{2,3,5\}} i}=\frac{\sum_{j\in\{2,3,5\}} c_j\cdot\prod_{i\neq j} i^{c_i}}{\prod_{i\in\{2,3,5\}} i^{c_i}}. \tag{22}$$

Verifiquemos la ecuación anterior. Si $c_i=0$, entonces el término i aparece en todos los sumandos y en el denominador, lo que significa que podemos cancelarlos. Para cancelar esta i en el producto, queremos que se conviertan en 1. Este es el caso de i^{c_i} ya que

$$i^{c_i}=\begin{cases} i & \text{si} \quad c_i=1 \\ 1 & \text{si} \quad c_i=0 \end{cases} \tag{23}$$

Para confirmar esto, revisemos las fracciones del ejemplo (22) y nuevamente tomemos $c_1=0$, $c_3=1$ y $c_5=0$.

$$\frac{\sum_{j\in\{2,3,5\}} c_j\cdot\prod_{i\neq j} i^{c_i}}{\prod_{i\in\{2,3,5\}} i^{c_i}}=\frac{c_2\cdot 3^{c_3}\cdot 5^{c_5}+c_3\cdot 2^{c_2}\cdot 5^{c_5}+c_5\cdot 2^{c_2}\cdot 3^{c_3}}{2^{c_2}\cdot 3^{c_3}\cdot 5^{c_5}\cdot}$$
$$=\frac{1\cdot 3^1\cdot 5^0+1\cdot 2^1\cdot 5^0+0\cdot 2^1\cdot 3^1}{2^1\cdot 3^1\cdot 5^0}$$
$$=\frac{1\cdot 3\cdot 1+1\cdot 2\cdot 1}{2\cdot 3\cdot 5}=\frac{1\cdot 3+1\cdot 2}{2\cdot 3}.$$

Usando esta lógica, podemos escribir $R_y(x)$ de (20) de la siguiente manera

$$R_y(x) = \sum_{i=0}^{n-1} \frac{e_i}{x - \alpha_i} = \frac{\sum_{j=0}^{n-1} e_j \cdot \prod_{i \neq j} (x - \alpha_i)^{e_i}}{\prod_{i=0}^{n-1} (x - \alpha_i)^{e_i}}. \tag{24}$$

Note que el denominador de $R_y(x)$ es ahora el polinomio localizador de errores, ya que

$$\sigma(x) = \prod_{i|e_i \neq 0} (x - \alpha_i) = \prod_{i=0}^{n-1} (x - \alpha_i)^{e_i}. \tag{25}$$

En realidad existe una conexión aún más directa entre $R_y(x)$ y el polinomio localizador de errores. Para mostrar esto, tomamos la derivada del polinomio localizador de errores. Como el polinomio localizador de errores es un producto de muchos términos $(x - \alpha_i)$, tenemos que aplicar la regla del producto. En general, la derivada de un producto con respecto a x es la siguiente.

$$\frac{d}{dx}\left[\prod_{i=1}^{k} f_i(x)\right] = \sum_{i=1}^{k} \left(\frac{d}{dx} f_i(x) \prod_{j \neq i} f_j(x)\right).$$

Si aplicamos esto a nuestro polinomio localizador de errores, entonces.

$$\frac{d}{dx}\left[\prod_{i=0}^{n-1} (x - \alpha_i)^{e_i}\right] = \sum_{i=0}^{n-1} \frac{d}{dx} \left((x - \alpha_i)^{e_i}\right) \prod_{j \neq i} (x - \alpha)^{e_i}. \tag{26}$$

En general, la derivada de $(x - \alpha_i)^{e_i}$ con respecto a x es

$$\frac{d}{dx}((x - \alpha_i)^{e_i}) = e_i \cdot (x - \alpha_i)^{e_i - 1} \cdot \frac{d}{dx}(x - \alpha_i) = e_i \cdot (x - \alpha_i)^{e_i - 1} \cdot 1.$$

Como todos los e_i son valores binarios, podemos hacer la siguiente restricción.

$$e_i \cdot (x-\alpha_i)^{e_i-1} = \begin{cases} 1 \cdot (x-\alpha_i)^0 = 1 & \text{si} \quad e_i = 1 \\ 0 \cdot (x-\alpha_i)^{-1} = 0 & \text{si} \quad e_i = 0. \end{cases} \tag{27}$$

Lo cual significa que

$$\frac{d}{dx}((x-\alpha_i)^{e_i}) = e_i.$$

Ahora podemos introducirlo a la ecuación (26) y obtenemos

$$\sum_{i=0}^{n-1} \frac{d}{dx}\left((x-\alpha_i)^{e_i}\right) \prod_{j\neq i}(x-\alpha)^{e_i} = \sum_{i=0}^{n-1} e_i \prod_{j\neq i}(x-\alpha_j)^{e_i}. \tag{28}$$

Casualmente, este es el numerador de nuestro $R_y(x)$. Esto significa que podemos escribir $R_y(x)$ en términos de nuestro polinomio localizador de errores.

$$R_y(x) = \sum_{i=0}^{n-1} \frac{e_i}{x-\alpha_i} = \frac{\sum_{j=0}^{n-1} e_j \cdot \prod_{i\neq j}(x-\alpha_i)^{e_i}}{\prod_{i=0}^{n-1}(x-\alpha_i)^{e_i}} = \frac{\frac{d}{dx}\sigma(x)}{\sigma(x)}.$$

Ahora usaremos un truco para mostrar cómo podemos calcular el polinomio localizador de errores $\sigma(x)$ solo a partir de $R_y(x)$. El primer paso es escribir $\sigma(x)$ en términos de potencias pares e impares de x, lo que se puede hacer para cualquier polinomio. Por ejemplo, tomemos $K(x) = x^7 + x^5 + x^4 + x^2 + x + 1$. Podemos escribir esto como

$$K(x) = (x^4 + x^2 + 1) + (x^7 + x^5 + x) = (x^4 + x^2 + 1) + x(x^6 + x^4 + x1).$$

Tengamos en cuenta que estamos trabajando con coeficientes binarios, por lo tanto, $(x+1)^2 = x^2 + 1^2$. Para los coeficientes de forma general $c, d \in \mathbb{F}_{2^m}$ se mantiene esto, $cx^2 + d = (\sqrt{c}x + \sqrt{d})^2$, donde $\sqrt{c}$ y $\sqrt{d}$ existen en $\mathbb{F}_{2^m}$, por el Teorema 1.1.3. Si todas las potencias son pares, podemos sacar la raíz cuadrada dividiendo todas las potencias por 2 y sacando raíces de los coeficientes. Esto significa que podemos

escribir $K(x)$ como:

$$K(x) = (x^4 + x^2 + 1) + x(x^6 + x^4 + x1) = (x^2 + x + 1)^2 + x(x^3 + x^2 + 1)^2 = A^2(x) + xB^2(x).$$

De esta forma podemos escribir cualquier polinomio en términos de $A(x)$ y $B(x)$.
Si tomamos la derivada de $K(x)$, obtenemos:

$$\begin{aligned}\frac{d}{dx}K(x) &= \frac{d}{dx}(A^2(x) + B^2(x)) \\ &= 2 \cdot A(x) \cdot \frac{d}{dx}A(x) + x \cdot \frac{d}{dx}B^2(x) + B^2(x) \cdot \frac{d}{dx}x \\ &\equiv 0 + 2 \cdot B(x) \cdot \frac{d}{dx}B(x) + B^2(x) \equiv B^2(x).\end{aligned}$$

Como esto es aplicable a cualquier polinomio, también podemos escribir $\sigma(x)$ en términos de $A(x)$ y $B(x)$ ordenados por potencias de x. Entonces

$$\sigma(x) = A^2(x) + xB^2(x) \text{ y } \frac{d}{dx}\sigma(x) = B^2(x) \tag{29}$$

Entonces nosotros podemos escribir $R_y(x)$ como

$$R_y(x) = \frac{\frac{d}{dx}\sigma(x)}{\sigma(x)} \equiv \frac{B^2(x)}{\sigma(x)} \rightarrow \sigma(x) \cdot R_y(x) \equiv B^2(x)$$

Podemos sustituir y reescribir en términos de $A(x)$ y $B(x)$.

$$\begin{aligned}\sigma(x) \cdot R_y(x) \equiv B^2(x) &\rightarrow (A^2(x) + xB^2(x))R_y(x) \equiv B^2(x) \\ &\rightarrow A^2(x) + xB^2(x) \equiv \frac{1}{R_y(x)}B^2(x) \rightarrow B^2(x)\left(x + \frac{1}{R_y(x)}\right) \equiv A^2(x) \\ &\rightarrow B(x)\sqrt{x + \frac{1}{R_y(x)}} \equiv A(x)\end{aligned}$$

Demos un paso atrás y hagamos una deducción sobre los grados de $A(x)$ y $B(x)$. En Primer lugar, se puede tener en cuenta que $\sigma(x)$ se puede escribir de la siguiente manera $\sigma(x) = A^2(x) + xB^2(x)$, además sabemos por hipótesis que se han introducido t errores o menos. A partir de la ecuación (25) nuestro polinomio localizador de errores tiene un grado de t o menor, ya que tomamos el producto de t elementos que contienen una x. Tengamos en cuenta que no debemos aplicar el módulo $g(x)$ en este polinomio, ya que esto reducirá el polinomio a un grado menor cuando hay t errores. Con lo dicho anteriormente se puede deducir que el grado de $A(x)$ es la mitad de la potencia par más alta en $\sigma(x)$ y el grado de $B(x)$ es la mitad del grado de la potencia impar más alta de $\sigma(x)$ menos 1, ya que dividimos las potencias impares de $\sigma(x)$ por x para obtener $B^2(x)$. Como el grado más alto posible es t, obtenemos que:

$$\text{E1 grado de } A(x) \leq \left\lfloor \frac{t}{2} \right\rfloor \text{ y el grado de } B(x) \leq \left\lfloor \frac{t-1}{2} \right\rfloor. \tag{30}$$

Si calculamos $v(x) = \sqrt{x + \frac{1}{R_y(x)}}$ entonces $B(x)v(x) \equiv A(x) \mod g(x)$ acorde a lo demostrado anteriormente. Note que $v(x) \in \mathbb{F}_{2^m}$ es la raíz cuadrada única del polinomio $x + \frac{1}{R_y(x)}$, es decir que $v(x)v(x) \equiv x + \frac{1}{R_y(x)} \mod g(x)$, por el Teorema 1.1.4.

Como nosotros conocemos a $R_y(x)$ y $g(x)$, podemos calcular $v(x)$ y usando el algoritmo extendido de Euclides en $v(x)$ y $g(x)$ de modo que obtengamos la siguiente ecuación en cada ejecución del algoritmo

$$A'(x) = B'(x)v(x) + H(x)g(x).$$

Continuando con el algoritmo hasta que los grados de $A'(x)$ y $B'(x)$ se ajuste a los grados mencionados en (30). Una vez que hayamos encontrado $A(x)$ y $B(x)$ correctas, podemos calcular el polinomio localizador de errores $\sigma(x)$ de acuerdo con la ecuación (29). Ahora aplicamos el polinomio localizador de errores y buscamos

los bits invertidos como se explico al comienzo de la sección.

El algoritmo de Patterson consiste en aplicar los siguientes pasos:

Algorithm 1: Algoritmo de Patterson

Input: El mensaje recibido y y el código de Goppa $\Gamma(L, g)$.
Output: Palabra código.

1 Calcular el polinomio $R_y(x)$
2 Calcular $T(x) = R_y(x)^{-1} \mod g(x)$.
3 Calcular $v(x) = \sqrt{T(x) + x} \mod g(x)$.
4 Calcular $A(x)$ y $B(x)$ con $A(x) = B(x)v(x) \mod g(x)$. Calcular el polinomio localizador de errores $\sigma(x) = A(x)^2 + xB(x)^2$.
5 Encontrar las raíces de $\sigma(x)$.
6 Encontrar el conjunto posiciones del vector de errores a partir de las raíces de $\sigma(x)$, es decir $E = \{i | e_i \neq 0\}$.
7 Definimos el vector de error $e = (e_1, \ldots, e_n)$ donde $e_i = 1$ si $i \in E$ o en caso contrario $e_i = 0$.
8 Definimos la palabra código $c = y - e$.

A pesar que quizás el algoritmo tenga apariencia sencilla, hay algunos conceptos claves en el entendimiento e implementación de este algoritmo. Estos conceptos incluyen el cálculo del síndrome, un método eficiente para calcular el inverso y las raíces cuadradas de un polinomio en un anillo cociente.

Ejemplo 4.5.1. *Vamos a trabajar en el campo $\mathbb{F}_{16}$ construido a partir del siguiente polinomio, $p(x) = x^4 + x + 1$. Ahora, hallaremos los códigos de Goppa con longitud $n = 16$, para ello tomaremos como vector $L = (\alpha, \alpha^2, \ldots, \alpha^{14}, 1, 0) \in \mathbb{F}_{16}^{16}$, siendo α un elemento primitivo de $\mathbb{F}_{16}$ y definimos el polinomio $g(x) = x^3 + \alpha^2 x + \alpha$, para tener que $g(l) \neq 0$ para todo $l \in L$.*

Para construir la matriz de paridad H del código, debemos calcular los elementos inversos de $g(l)$ donde $l \in L$. En este ejemplo, podemos expresar cada uno de los elementos menos el 0 como una cierta potencia de α. Como sabemos que $g(l) \neq 0$, entonces calcular los elementos inversos consiste en evaluar el polinomio $g(x)$ y dividirlo por $\alpha^{15} = 1$, así obtenemos los siguientes resultados:

1. $g(\alpha) = 2\alpha^3 + \alpha = \alpha$
2. $g(\alpha^2) = \alpha^6 + \alpha^4 + \alpha = \alpha^3 + \alpha^2 + 1 = \alpha^{13}$

3. $g(\alpha^3) = \alpha^9 + \alpha^5 + \alpha = \alpha^3 + \alpha^2 + \alpha = \alpha^{11}$

4. $g(\alpha^4) = \alpha^{12} + \alpha^6 + \alpha = 1$

5. $g(\alpha^5) = \alpha^{15} + \alpha^7 + \alpha = \alpha^3$

6. $g(\alpha^6) = \alpha^3 + \alpha^8 + \alpha = \alpha^3 + \alpha^2 + \alpha + 1 = \alpha^{12}$

7. $g(\alpha^7) = \alpha^6 + \alpha^9 + \alpha = \alpha^2$

8. $g(\alpha^8) = \alpha^9 + \alpha^{10} + \alpha = \alpha^3 + \alpha^2 + \alpha + 1 = \alpha^{12}$

9. $g(\alpha^9) = \alpha^{12} + \alpha^{11} + \alpha = \alpha + 1 = \alpha^4$

10. $g(\alpha^{10}) = \alpha^{12} + \alpha + 1 = \alpha^3 + \alpha^2 = \alpha^6$

11. $g(\alpha^{11}) = \alpha^{13} + \alpha^3 + \alpha = \alpha^2 + \alpha + 1 = \alpha^{10}$

12. $g(\alpha^{12}) = \alpha^6 + \alpha^{14} + \alpha = \alpha^2 + \alpha + 1 = \alpha^{10}$

13. $g(\alpha^{13}) = \alpha^9 + 1 + \alpha = \alpha^3 + 1 = \alpha^{14}$

14. $g(\alpha^{14}) = \alpha^{12}$

15. $g(1) = \alpha^2 + \alpha + 1 = \alpha^{10}$

16. $g(0) = \alpha$

Ahora, con los resultados anteriores y el teorema (4.3.6), se construye la matriz de paridad H, la cual tiene tamaño 3×16 y se ve de la siguiente forma:

$$H = \begin{pmatrix} \alpha^3+1 & \alpha^2 & \alpha+1 & \cdots & \alpha^3 & \alpha^2+\alpha & \alpha^3+1 \\ 1 & \alpha+1 & \alpha^3+\alpha+1 & \cdots & \alpha^2 & \alpha^2+\alpha & 0 \\ \alpha & \alpha^3+\alpha^2 & \alpha^2+\alpha+1 & \cdots & \alpha & \alpha^2+\alpha & 0 \end{pmatrix}. \tag{31}$$

Recordemos que los elementos en $\mathbb{F}_{16}$ los podemos representar como un vector en $\mathbb{F}_2^4$ el cual es una representación de una base. Si utilizamos $\{1, \alpha.\alpha^2, \alpha^3\}$, podemos por ejemplo representar el elemento $\alpha + 1$ como el vector $(1, 1, 0, 0)$. Por consiguiente la matriz H con entradas en $\mathbb{F}_2$ se define como:

$$H = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \end{pmatrix}. \tag{32}$$

Luego definimos la matriz generadora G de tamaño 4×16*, la cual es obtenida a partir de las operaciones elementales entre filas y columnas:*

$$G = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \end{pmatrix}. \tag{33}$$

De la matriz G se puede observar que los parámetros del código de Goppa son $[16, 4, 7]$*. Entonces el código* $\Gamma(g(x), L)$ *es capaz de corregir hasta* $t = 3$ *errores. Supongamos que Alice desea enviar a Bob el mensaje* $m = (0, 1, 0, 0, 1, 0, 0, 0)$*, utilizando el código* $\Gamma(g(x), L)$ *definido anteriormente. Recordando que la dimensión del código es* $k = 4$*, debemos dividir el mensaje en dos bloques de longitud* 4 *tal como se vio en la sección 4.1.*

Una vez dividido nuestro mensaje, pasamos a realizar la codificación utilizando la matriz generadora G. Así el mensaje codificado es:

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

Por tanto, suponiendo que se han producido $t = 3$ *errores en el primer bloque, podemos aplicar Patterson sobre este y así corregir los errores introducidos. Debemos mencionar que para el segundo bloque no es necesario corregirlo ya que el valor del síndrome es* 0*. Tomamos el vector de error de tamaño 16, donde los tres errores se encuentran en las coordenadas* $4, 8$ *y* 13*, es decir,*

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

entonces el vector que recibe Bob es:

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

Donde el primer y el segundo son respectivamente:

$$y_1 = (0,1,0,0,0,0,0,1,1,0,1,0,1,0,1,1)$$
$$y_2 = (1,0,0,0,1,0,1,0,1,1,0,0,0,0,1,1).$$

Aplicando el algoritmo de Patterson sobre y_1 *para encontrar las posiciones de error. Primero calculamos* $R_{y_1}(x)$*, cuyo valor es:*

$$Ry_1(x) = \sum_{i=0}^{15} \frac{y_{1_i}}{x - \alpha_i} \equiv (\alpha^3 + 1)x^2 + (\alpha^2 + \alpha)x + \alpha \mod g(x).$$

Ahora calculamos el polinomio $T(x)$*, el cual es el inverso de* $R_{y_1}(x) \mod g(x)$*:*

$$T(x) = (\alpha^2 + \alpha)x^2 + \alpha^3 x + \alpha^3 + \alpha.$$

Calculamos $v(x)$ de forma que $v^2(x) \equiv T(x) + x \mod g(x)$:

$$v(x) = (\alpha^3 + 1)x^2 + (\alpha^2 + \alpha)x + \alpha.$$

Ahora obtenemos los polinomios $A(x)$ y $B(x)$ de menor grado:

$$A(x) = (\alpha^2 + \alpha)x + 1$$
$$B(x) = \alpha x + \alpha^3 + \alpha + 1.$$

Por tanto el polinomio localizador de errores resultante es:

$$\sigma(x) = \alpha^2 x^3 + (\alpha^2 + \alpha + 1)x^2 + (\alpha^3 + 1)x + 1.$$

Finalmente calculamos las raíces del polinomio σ, las cuales corresponden a los valores de α^5, α^9 y α^{14} y obtenemos las posiciones en que ha ocurrido un error. Por último, realizando los cambios correspondientes podemos recuperar el mensaje $m = (0, 1, 0, 0, 1, 0, 0, 0)$.

5 | Criptografía basada en la teoría de códigos

Desde que apareció el concepto de criptografía a clave pública en 1977, la búsqueda de esquemas de identificación y criptosistemas seguros ha sido una de las áreas más activas en el campo de la criptografía. El desarrollo de algunos métodos de análisis de criptosistemas han demostrado que la mayoría de ellos son inseguros. Más de cuarenta años después del artículo fundamental de Deffie y Helman, los criptosistemas a clave pública dependen peligrosamente de sólo dos problemas: factorización de primos grandes y logaritmo discreto.

Decimos peligrosamente, ya que el algoritmo de Shor, (visto en el Capítulo 3) resuelve estos dos problemas y por lo tanto estos criptosistemas no serán seguros una vez lleguen los computadores cuánticos. La criptografía post-cuántica, se basa en encontrar otros tipos de problemas, los cuales no se puedan resolver en un computador clásico ni en uno cuántico. Como lo vimos en la Sección 2.3 existen varias familias de criptosistemas post-cuánticos, en esta sección estudiaremos los criptosistemas basados en teoría de códigos.

Robert J. McEliece en 1978 (un año después del criptosistema de Deffie y Helman) propuso un nuevo criptosistema a clave pública basado en la teoría de códigos algebraicos. A diferencia del RSA, las personas no lo adoptaron en la práctica debido al gran tamaño de la clave pública. El interés de los investigadores en el criptosistema de McEliece aumentó con la llegada de la computación cuántica, ya que a diferen-

cia de los sistemas basados en problemas de factorización de primos grandes y del logaritmo discreto, la seguridad del McEliece se basa en el problema de descifrado general el cual es *Np-Hard* y resiste los ataques de los computadores cuánticos [3].

En 1986, Niederreiter propone una variante inspirada en McEliece, en vez de utilizar la matriz generadora G utiliza la matriz de paridad H. Por lo anterior, la bibliografía reconoce al criptosistema de Niederreiter como el dual del criptosistema de McEliece. Aunque no podamos utilizar a McEliece en la práctica, debido al gran tamaño de su clave, este algoritmo ha inspirado a muchas variantes de criptosistemas basados en teoría de códigos, como por ejemplo [32], [41], [22], [2], [42], [31], [29].

En este capítulo, proporcionamos una descripción general del criptosistema de McEliece en su forma original [30]. Comenzamos con la descripción general del sistema original y luego explicaremos el criptosistema de Neiderreiter. Finalmente daremos una explicación sobre la versión del criptosistema de McEliece propuesto a la competencia de la NIST, el cual es una de las finalistas.

5.1. Criptosistemas de McEliece y Neiderreiter

Criptosistema de McEliece

La idea general del criptosistema es seleccionar un código (que tenga un algoritmo de decodificación eficiente asociado) y disfrazar el código como un código lineal aleatorio. Dado que el problema de descifrar un código lineal arbitrario es *NP-Hard*, podemos usar la descripción del código como parte de la clave privada, mientras que el código disfrazado puede servir como clave pública.

Sean $m, t \in \mathbb{N}$, para cada polinomio irreducible $g(x)$ sobre el campo $\mathbb{F}_{2^m}$ de grado t y el conjunto de elementos L definidos como se menciona en la Definición 4.3.6, existe un código de Goppa binario $\Gamma(L, g)$ irreducible de tamaño $n = 2^m$ y de

dimensión $k \geq n - mt$, capaz de corregir a lo sumo t errores. Como hemos mencionado en la sección 4.3.5, los códigos de Goppa son lineales lo cual permite describir la matriz generadora G de tamaño $k \times n$ y sabemos que existe un algoritmo de decodificación eficiente para corregir a lo sumo t errores.

A continuación vamos a dar una descripción formal del criptosistema.

Generación de claves

El objetivo de esta etapa consiste en generar el conjunto de llaves que serán usadas para las etapas de cifrado y descifrado.

1. Alice selecciona un código de Goppa binario $\Gamma(L, g)$ capaz de corregir t errores.
2. Alice genera una matriz G de tamaño $k \times n$ del código $\Gamma(n, k)$.
3. Selecciona una matriz binaria no singular S de tamaño $k \times k$.
4. Selecciona una matriz de permutación P de tamaño $n \times n$.
5. Calcular la matriz $\hat{G} = S \times G \times P$.
6. La clave pública de Alice es $(\hat{G}, t)$ y la clave privada es (S, G, P).

Nota: Podemos observar que $\hat{G}$ hace referencia a la matriz generadora del código disfrazado como lo hemos mencionado en la introducción de esta capítulo.

Cifrado

Supongamos que Bob quiere enviarle un mensaje m de tamaño k a Alice utilizando la llave pública $(\hat{G}, t)$, entonces Bob debe realizar los siguientes pasos:

1. Generar un vector binario aleatorio e de tamaño n tal que $wt(e) \leq t$.
2. Calcular el texto cifrado de la siguiente manera $c = m \times \hat{G} \oplus e$.

Descifrado

Alice recibe el mensaje de Bob y para recuperar el mensaje original, Alice debe

realizar los siguientes pasos:

1. Calcular $c' = c \times P^{-1} = (m\hat{G}P^{-1}) \oplus eP^{-1} = mSGPP^{-1} \oplus eP^{-1} = mSG \oplus eP^{-1}$.

2. Usar el algoritmo de descifrado para el código de Goppa $\Gamma(L, g)$ para encontrar $m' = m \times S$, ya que $wt(eP^{-1}) \leq t$.

3. Calcular $m = m' \times S^{-1}$.

Ejemplo 5.1.1. *Para mostrar el funcionamiento del criptosistema, se desarrollo un código en SAGE el cual podemos ver en el siguiente link:* `https://github.com/dfrafelipe30/Tesis.git`

Criptosistema Niederreiter

En 1986, Niederreiter propuso un tipo de criptosistema a clave publica basado en códigos de corrección de errores [32]. La idea original es utilizar los códigos *GRS* (Sección 4.3.3) en su criptosistema, lo cual permite tamaños de claves más pequeños que los del criptosistema de McEliece. Sin embargo, en 1992 Sidelnikov y Shestakov atacaron el criptosistema de Niederreiter usando los códigos *GRS*. Por otro lado, se demostró que utilizando los códigos de Goppa, esta propuesta tenía una seguridad equivalente a la propuesta de McEliece [26]. La diferencia entre los dos criptosistemas, se encuentra en no cifrar el mensaje m directamente, sino codificar el mensaje en el vector de error por medio de la función $\phi_{n,t} : \{0,1\}^l \to W_{2,n,t}$, donde $W_{2,n,t} = \{c \in \mathbb{F}_2^n : wt(c) = t\}$ y $l = \log_2 \lfloor \binom{n}{t} \rfloor$. En el Algoritmo 2 podemos ver cómo se construye la función $\phi_{n,t}$. Este algoritmo tiene complejidad $O(n^2 \log_2 n)$ y su inversa es fácil de definir:

$$\phi_{n,t}^{-1}(e) = \sum_{i=1}^{n} e_i \times \binom{i}{\sum_{j=0}^{i} e_j}$$

Al representar el mensaje como el vector de error, obtenemos la variante dual del criptosistema de McEliece.

Algorithm 2: $\phi_{n,t} : \{0,1\}^l \to W_{2,n,t}$

Input: $x \in \{0,1\}^l$

Output: Una palabra $e = (e_1, e_2, \cdots, e_n)$ de peso t y tamaño n

$a = \binom{n}{t}$

$c' = 0$, $j = n$ y $t' = t$

$i =$ el número natural que representa x en base decimal

while $j > 0$ **do**

  $c' = a \times \frac{j-t'}{j}$

  if $i < c'$ **then**

    $e_j = 0$ $a = c'$

  else

    $e_j = 1$ $i = i - c'$ $a = a \times \frac{t'}{j}$ $t' = t' - 1$

  $j = j - 1$

Para describir el criptosistema, debemos seleccionar un código de Goppa $\Gamma(L, g)$ sobre el campo $\mathbb{F}_q$ capaz de corregir t errores y una matriz de paridad H de tamaño $(n-k) \times n$ indistinguible de una matriz aleatoria. Generar una matriz M no singular de tamaño $(n-k) \times (n-k)$ y P una matriz de permutación de tamaño $n \times n$, todo sobre el campo $\mathbb{F}_q$. El criptosistema se define entonces:

1. *Llave privada:* H, M, P.
2. *Llave pública:* $H' = M \times H \times P$ y t.
3. *Mensaje:* vector m de tamaño n sobre el campo $\mathbb{F}_q$ y $wt(m) \leq t$.
4. *Cifrado:* Un mensaje m es representado como un vector $e \in \{0,1\}^n$ de peso t. Para cifrar calculamos $c = e \times H'^T$, note que c de dimensión $n - k$.
5. *Descifrado:* Ya que $c = e \times (M \times H \times P)^T$, entonces $c(M^T)^{-1} = (e \times P^T) \times H^T$, usamos un algoritmo de descifrado para $\Gamma(L, g)$ para encontrar $c \times P^T$ y obtener e.

Nivel de seguridad	(n, k)	t	McEliece	RSA
80 bits	$(2048, 1751)$	27	520047	1248
128 bits	$(2960, 2288)$	56	1537536	3248
256 bits	$(6624, 5129)$	115	7667855	15424

Cuadro 5.1: Parámetros para varios niveles de seguridad en variantes de CCA2 del criptosistema McEliece.

Ventajas de los criptosistemas

Hay dos principales ventajas de estos criptosistemas que pueden ser inferidas. La primera es la eficiencia con la que se puede implementar los algoritmos de cifrado y descifrado, ya que los códigos de Goppa tienen un algoritmo rápido para el descifrado, como lo hemos visto en la Sección 4.4. La segunda ventaja que se puede observar involucra el problema de decodificación del mensaje con t errores en un código lineal, ya que se demostró que es *NP-Hard*. Sin embargo, usando los códigos de Goppa resulta que el tamaño de la clave es tan grande que dificulta su uso en actividades diarias.

La ventaja del criptosistema de Niederreiter es el tamaño de la clave pública, ya que para guardar la clave es suficiente almacenar únicamente la parte redundante de la matriz H. La desventaja es el hecho de que el mapeo $\phi_{n,t}$ ralentiza el cifrado y el descifrado.

Seguridad de los criptosistemas

Como se mencionó anteriormente, se demostró que el criptosistema de Niederreiter (utilizando los códigos de Goppa) tiene una seguridad equivalente al criptosistema de McEliece.

Los ataques al criptosistema del McEliece se dividen principalmente en dos tipos, ataques estructurales y ataques de decodificación.

El primer tipo de ataque, conocido como ataque estructural, intenta recuperar la clave secreta mediante el estudio del código $\hat{G}$. Actualmente no existen ataques significativos para McEliece, sin embargo existen ataques estructurales a variantes del McEliece como [38], [28], [17], [43], [15], [33], [6], [25], [7], [8].

El segundo tipo de ataque, busca recuperar el mensaje m dado un texto cifrado, sin embargo este tipo de ataque presenta una dificultad de costo computacional, debido a que el atacante debe estar recolectando los textos cifrados que se envían para así conocer la estructura del mensaje, ya que no se conoce la clave secreta. Adicionalmente este tipo de ataque tampoco representa un riesgo para la seguridad del criptosistema de McEliece.

Faugère et al. [14] presentaron un distintor para el criptosistema de McEliece para ciertos parámetros de los códigos Goppa. Si bien el distintor funciona solo en un caso especial y no afecta la seguridad del esquema general si es la fuente de varios ataques de variantes de McEliece tales como [18].

Hay un tipo de ataque denominado *ataque adaptativo de texto cifrado elegido* ($CCA2$ - *Adaptive Chosen-ciphertext attack*), en donde el atacante tiene acceso a un oráculo, al cual le puede pedir que descifre los textos cifrados elegidos por él, durante un tiempo limitado. Decimos que un criptosistema es seguro contra este tipo de ataque, si dicho atacante no tiene ninguna ventaja para descifrar un texto cifrado objetivo bajo estas circunstancias.

El criptosistema de McEliece es vulnerable a este tipo de ataques, ya que si el atacante quiere obtener información de m a partir de $c = m\hat{G} + e$, le puede pedir al oráculo que descifre $c' = c + m'\hat{G} = (m + m')\hat{G} + e$. El oráculo le va a devolver $m + m'$, por lo que podrá recuperar m. Sin embargo, en el 2001 Imai y Kobara propusieron [24] una variante de McEliece resistentes a los ataques de tipo $CCA2$.

Debemos resaltar que hasta el momento no existe un ataque que rompa el criptosistema de McEliece para todos los parámetros ya definidos, sin embargo debido al gran tamaño su clave, no se utiliza en la práctica.

5.2. Classic McEliece

En *Classic McEliece: conservative code-based cryptography* [4], Berstein *et al.* proponen un mecanismo que permite comunicarse de forma segura basada en teoría de códigos. El artículo fue sometido a la competencia de la NIST y actualmente es uno de los finalistas. No es un criptosistema como tal, los autores proponen un mecanismo para compartir una clave secreta de forma segura, de tal manera que los usuarios se puedan comunicar a través de criptosistemas simétricos utilizando esta clave.

La propuesta se basa en un mecanismo que llamamos “encapsulamiento de claves", el cual es llamado KEM por sus siglas en ingles (Key Encapsulation Mechanism) y consta de tres partes: generación de claves, encapsulamiento y desencapsulamiento. Miremos cada uno de estos tres algoritmos a continuación:

Supongamos que Alice le pide a Bob que establezcan una clave de sesión utilizando el mecanismo de encapsulamiento del *Classic McEliece* en el campo $\mathbb{F}_{2^m}$.

Generación de claves

La parte de generación de claves es un algoritmo probabilístico que toma como entrada un parámetro de seguridad (como por ejemplo la seguridad requerida en bits) y genera una clave pública p_k y una clave privada s_k.

Bob genera su par de claves del *Classic McEliece* de la siguiente manera:

1. Generar un polinomio irreducible aleatorio $g(x) \in \mathbb{F}_{2^m}[x]$ de grado t.
2. Seleccionar uniformemente el conjunto $L = \{\alpha_1, \ldots, \alpha_n\} \subset \mathbb{F}_{2^m}$ donde todos los elementos son diferentes y $g(\alpha_i) \neq 0$ para todo $i \in [1, n]$.
3. Calcular la matriz $\hat{H} = \{h_{i,j}\}$ donde $h_{i,j} = \alpha_j^{i-1} g(\alpha_i)^{-1}$ para $i \in [1, t]$ y $j \in [1, n]$, *i.e.* la matriz de paridad para el código $\Gamma(g(x), L)$.
4. Reemplazar cada entrada de la matriz $\hat{H}$ (cuyos elementos están definidos en $\mathbb{F}_{2^m}$) por vectores (columna) en $\mathbb{F}_2^m$ usando el isomorfismo entre $\mathbb{F}_{2^m} \to \mathbb{F}_2^m$ y obtener la matriz $\tilde{H}$ de tamaño $mt \times n$.

5. Aplicar eliminación Gaussiana en $\tilde{H}$ para tratar de obtener una matriz sistemática de la forma $H = (I_{mt}|T_{mt\times(n-mt)})$. Si no es posible, Bob vuelve al paso 1. de este algoritmo.

6. Generar una cadena s de n bits aleatoriamente uniforme.

7. La Clave pública de Bob es $p_k = T$ y clave privada es $s_k = (s, g(x), L)$.

Algoritmo de encapsulamiento

Alice genera una clave de sesión K y un texto cifrado C de la siguiente manera:

1. Generar un vector aleatorio $e \in \mathbb{F}_2^m$ con peso de Hamming t de la misma manera que en el algoritmo de Niederreiter visto en la Sección 5.1.

2. **Cifrado Niederreiter:** utilizando la clave pública $p_k = T$ de Bob, Alice recupera $H = (I|T)$ y genera un vector $c_0 = He = (I|T)e$ de longitud mt en $\mathbb{F}_2^{mt}$.

3. Sea **HASH** la función de hash SHAKE-256 y 2 el valor inicial representado en un byte. Definimos $c_1 = \mathbf{HASH}(2, e)$ y obtenemos así $C = (c_0, c_1)$ de tamaño $mt + 256$ bits.

4. Calcular una clave de sesión $K = \mathbf{HASH}(1, e, C)$, donde 1 es el valor inicial representado en un byte.

Algoritmo de desencapsulamiento

Bob recibe el texto cifrado $C = (c_0, c_1)$ de Alice y generar la clave de sesión K'. Recuerde que $c_0 \in \mathbb{F}_2^{mt}$ y $c_1 \in F_2^{256}$ y que además Bob posee la clave privada $(s, g(x), L)$.

1. Definir $v = (c_0, 0, \ldots, 0) \in \mathbb{F}_2^n$ agregando $n - mt$ ceros.

2. Asignar $b = 1$.

3. Usando el algoritmo de descifrado del criptosistema de Niederreiter, tratamos de encontrar la palabra código c en el código de Goppa $\Gamma(L, g(x))$ tal que

$d(c, v) \leq t$. Si no existe la palabra código retornamos $\perp$.

4. En caso de que exista la palabra código c, definimos el vector $e = v + c$. Si $wt(e) = t$ y $c_0 = He$ se devuelve e, de lo contrario se devuelve $\perp$.

5. Si el descifrado retorna $\perp$, se define a $e = s$ y $b = 0$.

6. Calcular $c_1' = \mathbf{HASH}(2, e)$ y revisamos si $c_1' = c_1$. Si no se cumple la anterior condición, asignamos $e = s$ y $b = 0$.

7. Se calcula la clave de la sesión $K' = \mathbf{HASH}(b, e, C)$.

Note que si no hay falla en ninguna de las etapas del proceso de desencapsulamiento y $c_1' = c_1$, entonces seguramente la clave de sesión K' sera idéntica a K. De manera equivalente si Bob recibe un texto cifrado válido, es decir $C = (c_0, c_1)$ con el valor de $c_0 = He$ para algún vector $e \in \mathbb{F}_2^n$ de peso de Hamming t y $c_1 = \mathbf{HASH}(2, e)$, el descifrado siempre resulta en encontrar el vector e. En este escenario, se estable la misma clave de sesión.

Parámetros propuestos

Los autores del *Classic McEliece* proponen los siguientes parámetros que generan diferentes niveles de seguridad.

Variante	n	m	t	$k = n - mt$	Seguridad	Polinomio
mceliece6960119	6960	13	119	5413	128	$g(x) = x^{13} + x^4 + x^3 + x + 1$
mceliece8192128	8192	13	128	6528	256	$g(x) = x^{13} + x^4 + x^3 + x + 1$

Seguridad del *Classic McEliece*

En el artículo los autores presentan un análisis de la seguridad del *Classical McEliece*, comparando con los ataques del cripstosistema de McEliece y sus diferentes variantes. Adicionalmente en estos casi cuatro años de competencia no ha existido un ataque relevante para esta propuesta. En esta tesis nos enfocaremos en la demostración de por qué esta es una variante $IND - CCA2$ segura. La seguridad de tipo $CCA2$ ya fue presentada en la sección anterior, definamos ahora dos nociones de seguridad que necesitamos para entender la seguridad $IND - CCA2$.

Definición 5.2.1. *Un criptosistema tiene la propiedad de indistiguibilidad (denotada IND), si el atacante no puede ganar el siguiente juego. Hay dos jugadores, el retador (R) y el atacante (A).*

1. *El retador R debe escoger una pareja de claves pública y secreta (p_k, s_k) y revelar p_k al atacante.*
2. *El atacante A genera dos textos planos de mismo tamaño m_0 y m_1.*
3. *El retador selecciona al azar un $b \in \{0, 1\}$ y envía al atacante $c^* = Enc_{pk}(m_b)$.*
4. *El atacante gana si luego de un número polinomial de operaciones puede deducir correctamente b.*

En el modelo de ataques más fuerte visto en la sección anterior llamado $CCA2$, para encontrar el mensaje original m tal que $c = Enc(m)$, el atacante puede hacer uso de un oráculo de descifrado durante el ataque, con la única excepción de que no puede pedir el descifrado de c. Veamos entonces cómo uniendo estas dos nociones de seguridad se puede definir la propiedad $IND - CCA2$.

Definición 5.2.2. *El criptosistema es $IND - CCA2$ si:*

1. *El retador R debe escoger una pareja de claves pública y secreta (p_k, s_k) y revelar p_k al atacante.*
2. *El atacante A puede hacer una o varias consultas al oráculo pidiendo descifrar unos textos cifrados que el escoge.*
3. *El atacante A genera dos textos planos de mismo tamaño m_0 y m_1.*
4. *El retador selecciona al azar un $b \in \{0, 1\}$ y envía al atacante $c^* = Enc_{pk}(m_b)$.*
5. *El atacante A puede hacer una o varias consultas al oráculo pidiendo descifrar unos textos cifrados que él escoge (a excepción de c^*).*
6. *El atacante gana si luego de un número polinomial de operaciones puede deducir correctamente b.*

Decimos que un criptosistema es $IND-CCA2$ si la probabilidad del atacante de ganar es ligeramente superior a $1/2$ ya que es la misma probabilidad de elegir aleatoriamente el bit b.

Teniendo en cuenta la definición de $IND-CCA2$, podemos ver por que el *Classic McEliece* es resistente a este tipo de seguridad. De acuerdo con los autores, la demostración se basa en las siguientes cuatro características de la propuesta.

1. La clave de sesión proviene del hash del vector de entrada aleatorio uniforme e.
2. El texto cifrado también consta de confirmación, es decir, otro hash del vector e.
3. Después del cálculo de e, utilizando la clave privada del KEM, el texto cifrado se vuelve a calcular para confirmar que coincide.
4. Si el descifrado falla para el cálculo inverso, KEM no devuelve el error; en su lugar, devuelve una función pseudoaleatoria del texto cifrado, específicamente un hash criptográfico sofisticado, una clave privada separada y el texto cifrado.

Las tres primeras se heredan del artículo de DEN [10] y la última del artículo [37].

Classic McEliece es el único criptosistema basado en teoría de códigos que actualmente es finalista en la competencia de la NIST. Sin embargo, existen otras opciones, como lo es *BIKE: Bit-Flipping Key Encapsulation*, el cual consiste en un mecanismo de entregas de claves basado también en teoría de códigos. Como pudimos evidenciar esta es un área muy interesante para estudiar diferentes criptosistemas. Finalmente es importante hacer énfasis en el gran potencial que tienen estos criptosistema y además las mejoras que pueden surgir a partir de ellos como nuevas variantes. En cuanto a la computación cuántica, parece no haber ningún algoritmo conocido que pueda romper este criptosistema con facilidad, por lo que en un futuro podemos

ver implementaciones en el mundo de la seguridad informática.

Bibliografía

[1] P. Benioff. The computer as a physical system: A microscopic quantum mechanical Hamiltonian model of computers as represented by Turing machines. *Journal of statistical physics*, 22:563–591, 1980.

[2] T. P. Berger and P. Loidreau. How to mask the structure of codes for a cryptographic use. *Designs, Codes and Cryptography*, 35(1):63–79, 2005.

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

[4] D. J. Bernstein, T. Chou, T. Lange, I. V. Maurich, R. Misoczki, R. Niederhagen, E. Persichetti, C. Peters, P. Schwabe, N. Sendrier, J. Szefer, and W. Wang. Post-quantum cryptography. *NIST PQC Competition*, 2017.

[5] P. J. Cameron. *Introduction to algebra*. Oxford University Press on Demand, 2008.

[6] A. Couvreur, P. Gaborit, V. Gauthier-Umaña, A. Otmani, and J.-P. Tillich. Distinguisher-based attacks on public-key cryptosystems using Reed–Solomon codes. *Designs, Codes and Cryptography*, 73(2):641–666, 2014.

[7] A. Couvreur, I. Márquez-Corbella, and R. Pellikaan. A polynomial time attack against algebraic geometry code based public key cryptosystems. In *2014 IEEE International Symposium on Information Theory*, pages 1446–1450. IEEE, 2014.

[8] A. Couvreur, A. Otmani, J.-P. Tillich, and V. Gauthier–Umaña. A polynomial-time attack on the BBCRS scheme. In *IACR International Workshop on Public Key Cryptography*, pages 175–193. Springer, 2015.

[9] D. Das, U. Lanjewar, and S. Sharma. The art of cryptology: From ancient number system to strange number system. *International Journal of Application or Innovation in Engineering & Management (IJAIEM)*, 2(4), 2013.

[10] A. W. Dent. A designer's guide to KEMs. In *IMA International Conference on Cryptography and Coding*, pages 133–151. Springer, 2003.

[11] D. Deutsch. Quantum theory, the Church–Turing principle and the universal quantum computer. *Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences*, 400:97–117, 1985.

[12] D. E. Deutsch. Quantum computational networks. *Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences*, 425:73–90, 1989.

[13] D. Engelbert, R. Overbeck, and A. Schmidt. A summary of McEliece type cryptosystems and their security. *Journal of Mathematical Cryptology*, pages 151–199, 2007.

[14] J.-C. Faugère, V. Gauthier-Umaña, A. Otmani, L. Perret, and J.-P. Tillich. A distinguisher for high-rate McEliece cryptosystems. *IEEE Transactions on Information Theory*, 59(10):6830–6844, 2013.

[15] C. Faure and L. Minder. Cryptanalysis of the mceliece cryptosystem over hyperelliptic codes. In *Proceedings of the 11th international workshop on Algebraic and Combinatorial Coding Theory, ACCT*, pages 99–107, 2008.

[16] R. P. Feynman. Simulating physics with computers. *International Journal of Theoretical Physics*, pages 467–488, 1982.

[17] M. Finiasz and N. Sendrier. Security bounds for the design of code-based cryptosystems. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 88–105. Springer, 2009.

[18] V. Gauthier–Umaña, A. Otmani, and J.-P. Tillich. A distinguisher-based attack of a homomorphic encryption scheme relying on Reed-Solomon codes. *arXiv preprint arXiv:1203.6686*, 2012.

[19] V. D. Goppa. A new class of linear correcting codes. *Problemy Peredachi Informatsii*, 6(3):24–30, 1970.

[20] W. C. Huffman and V. Pless. *Fundamentals of error-correcting codes*. Cambridge university press, 2010.

[21] IBM. First Demonstration of Shor's Historic Factoring Algorithm. `https://www-03.ibm.com/press/us/en/pressrelease/965.wss`, 2001.

[22] H. Janwa and O. Moreno. McEliece public key cryptosystems using algebraic-geometric codes. *Designs, Codes and Cryptography*, 8(3):293–307, 1996.

[23] J. Justesen and T. Høholdt. *A course in error-correcting codes*, volume 1. European Mathematical Society, 2004.

[24] K. Kobara and H. Imai. Semantically secure mceliece public-key cryptosystems-conversions for McEliece PKC. In *International Workshop on Public Key Cryptography*, pages 19–35. Springer, 2001.

[25] G. Landais and J.-P. Tillich. An efficient attack of a McEliece cryptosystem variant based on convolutional codes. In *International Workshop on Post-Quantum Cryptography*, pages 102–117. Springer, 2013.

[26] Y. X. Li, R. H. Deng, and X. M. Wang. On the equivalence of McEliece's and Niederreiter's public-key cryptosystems. *IEEE Transactions on Information Theory*, 40(1):271–273, 1994.

[27] S. Ling and C. Xing. *Coding theory: a first course*. Cambridge University Press, 2004.

[28] P. Loidreau and N. Sendrier. Weak keys in the McEliece public-key cryptosystem. *IEEE Transactions on Information Theory*, 47(3):1207–1211, 2001.

[29] C. Löndahl and T. Johansson. A new version of McEliece pkc based on convolutional codes. In *International Conference on Information and Communications Security*, pages 461–470. Springer, 2012.

[30] R. J. McEliece. A public-key cryptosystem based on algebraic coding theory. *JPL DSN Progress Report*, pages 114–116, 1978.

[31] R. Misoczki, J.-P. Tillich, N. Sendrier, and P. S. Barreto. MDPC-McEliece: New McEliece variants from moderate density parity-check codes. In *2013 IEEE international symposium on information theory*, pages 2069–2073. IEEE, 2013.

[32] H. Niederreiter. Knapsack-type cryptosystems and algebraic coding theory. *Prob. Contr. Inform. Theory*, 15(2):157–166, 1986.

[33] A. Otmani, J.-P. Tillich, and L. Dallot. Cryptanalysis of two McEliece cryptosystems based on quasi-cyclic codes. *Mathematics in Computer Science*, 3(2):129–140, 2010.

[34] N. Patterson. The algebraic decoding of Goppa codes. *IEEE Transactions on Information Theory*, 21(2):203–207, 1975.

[35] I. S. Reed and G. Solomon. Polynomial codes over certain finite fields. *Journal of the society for industrial and applied mathematics*, 8:300–304, 1960.

[36] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[37] T. Saito, K. Xagawa, and T. Yamakawa. Tightly-secure key-encapsulation mechanism in the quantum random oracle model. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 520–551. Springer, 2018.

[38] N. Sendrier. Finding the permutation between equivalent linear codes: The support splitting algorithm. *IEEE Transactions on Information Theory*, 46(4):1193–1203, 2000.

[39] C. E. Shannon. A mathematical theory of communication. *Bell system technical journal*, 27:379–423, 1948.

[40] P. W. Shor. Algorithms for quantum computation: discrete logarithms and factoring. In *Proceedings 35th annual symposium on foundations of computer science*, pages 124–134. Ieee, 1994.

[41] V. M. Sidel'nikov. Open coding based on Reed–Muller binary codes. *Diskretnaya matematika*, 6(2):3–20, 1994.

[42] V. M. Sidelnikov. A public-key cryptosystem based on binary Reed-Muller codes. *Discrete Math. Apli.*, 4(3):191–207, 1994.

[43] V. M. Sidelnikov and S. O. Shestakov. On insecurity of cryptosystems based on generalized Reed-Solomon codes. 1(4):439–444, 1992.

[44] D. R. Simon. On the power of quantum computation. *SIAM journal on computing*, 26(5):1474–1483, 1997.

[45] D. R. Stinson and M. Paterson. *Cryptography: theory and practice*. CRC press, 2018.

[46] V. G. Umaña. Post-quantum cryptography. 2011.

[47] S. A. Vanstone and P. C. Van Oorschot. *An introduction to error correcting codes with applications*, volume 71. Springer Science & Business Media, 2013.

A1 | Apéndice

Gran parte del álgebra abstracta implica propiedades de enteros. En esta sección recopilamos las propiedades que necesitamos para referencias futuras.

Definición A1.0.1. *El máximo común divisor de dos enteros $a, b \in \mathbb{Z}$ diferentes de cero, es el mayor divisor común de a y b. Denotado como $gcd(a, b)$. Cuando el $gcd(a, b) = 1$, decimos que a y b son primos relativos.*

Definiremos una propiedad fundamental de los enteros la cual es usada con mucha frecuencia.

Lema A1.0.1. *Sea $a, b \in \mathbb{Z}$ con $b > 0$, entonces $gcd(0, b) = b$.*

Lema A1.0.2. *Sea c y d enteros diferentes de cero. Si q y r son enteros tal que $c = d \times q + r$, entonces el $gcd(c, d) = gcd(d, r)$.*

Teorema A1.0.1. *Sea $a, b \in \mathbb{Z}$ con $a > b \geq 0$. Entonces el $gcd(a, b)$ es un número natural d tal que*

1. *d divide a y b.*
2. *Si k es un entero que divide a a y b, entonces k divide d.*

Teorema A1.0.2. *Sea $a, b \in \mathbb{Z}$ diferentes de cero. Entonces existen enteros x y r tal que $gcd(a, b) = ax + by$.*