Universidad de Valladolid
Facultad de Ciencias

# Trabajo Fin de Grado

Grado en Matemáticas

**PIR: Recuperación de Información de forma Privada**

***Autor: Sergio Díez García***

***Tutor: Diego Ruano Benito***

# Agradecimientos

A Diego Ruano y Jose Ignacio Farrán, tutores de los dos trabajos fin de grado por ayudarme durante todo el año en la elaboración de los mismos.

# Índice general

**0. Introduction** **1**

Objetivos . . . . . 4

**1. Teoría de la información y Códigos Correctores Lineales** **6**

1.1. Teoría de la información . . . . . 6

1.2. Códigos lineales . . . . . 9

1.3. Matriz Generadora y Codificación Sistemática . . . . . 10

1.4. Matriz de Control y Código Dual . . . . . 12

1.5. Decodificación . . . . . 14

1.6. Códigos MDS . . . . . 17

**2. Modelo PIR y primeros protocolos** **23**

2.1. Modelo . . . . . 24

2.2. Protocolo PIR para b=1 . . . . . 27

2.3. Protocolo PIR para d-1 servidores cooperantes . . . . . 31

2.4. Otro protocolo PIR más eficiente . . . . . 36

**3. El producto estrella y su utilidad en los protocolos PIR** **40**

3.1. Producto estrella . . . . . 40

3.2. Protocolo . . . . . 42

3.3. Particularidades en los códigos GRS . . . . . 47

3.3.1. Un ejemplo en SAGE . . . . . 48

**4. PIR con servidores bizantinos y no responsivos** **55**

4.1. Protocolo inicial . . . . . 55

4.2. Particularidades con los códigos GRS . . . . . . . . . . . . . . . . . . . . . . . 59
4.2.1. Un ejemplo en Sage para servidores bizantinos y no responsivos . . . 60
4.3. Versión mejorada del protocolo para servidores bizantinos y no responsivos 63
4.3.1. Ejemplo en Sage . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 69

**A. Implementación de los protocolos PIR** **75**

A.1. Programas generales . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 75
A.1.1. Codificación de archivos en base de datos . . . . . . . . . . . . . . . 75
A.1.2. Cálculo de las respuestas por parte del servidor . . . . . . . . . . . . 77
A.1.3. Cálculo de las respuestas con ruido . . . . . . . . . . . . . . . . . . . 77
A.1.4. Recuperación del archivo original a partir del archivo en forma matricial . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 78
A.1.5. Producto estrella de dos códigos . . . . . . . . . . . . . . . . . . . . 79
A.2. Protocolos del capítulo 2 . . . . . . . . . . . . . . . . . . . . . . . . . . . 80
A.2.1. Creaciones de matrices solicitud . . . . . . . . . . . . . . . . . . . . 80
A.2.2. Recuperación del archivo a partir de las respuestas . . . . . . . . . . 84
A.3. Protocolo para códigos generales . . . . . . . . . . . . . . . . . . . . . . . 87
A.3.1. Creación de matrices solicitud . . . . . . . . . . . . . . . . . . . . . . 87
A.3.2. Recuperación del archivo a partir de las respuestas . . . . . . . . . . 89
A.3.3. Obtención de los códigos D y Star para el caso GRS . . . . . . . . . 91
A.4. Protocolo inicial para servidores bizantinos y no responsivos . . . . . . . . . 92
A.4.1. Creación de matrices solicitud . . . . . . . . . . . . . . . . . . . . . . 92
A.4.2. Recuperación del archvio a partir de las respuestas . . . . . . . . . . 93
A.5. Protocolo para servidores bizantinos y no responsivos mejorado . . . . . . . 94
A.5.1. Creación de las matrices solicitud . . . . . . . . . . . . . . . . . . . . 94
A.5.2. Recuperación del archivo a partir de las respuestas . . . . . . . . . . 95

# Introducción

En la actualidad, el acceso a información almacenada en bases de datos es algo que realizamos continuamente cuando, por ejemplo, estamos navegando en internet, o accediendo a cualquier servicio on-line. Como es lógico, existen gran cantidad de medidas de seguridad para garantizar que estos procedimientos se realicen de manera segura.

Entre ellas, se usan herramientas para protegernos contra una posible tercera persona que esté "escuchando" nuestras operaciones con el fin de robar nuestros datos. También las bases de datos cuentan con sistemas de autorización y autenticación para restringir el acceso a parte de los datos almacenados en su base de datos, de forma que sólo las personas autorizadas para ello sean capaces de extraer cierta información. Sin embargo, en el caso de proteger al propio usuario contra el propietario de la base de datos no se toma ninguna medida de seguridad.

Existen muchos casos donde este nivel de seguridad podría ser realmente útil. Un primer ejemplo podría ser el de un régimen opresor, donde que un usuario consulte cierta información sensible hace que dicho usuario sea añadido a una lista negra. Otro ejemplo podría ser el de una empresa que esté considerando la adquisición de otra más pequeña. Para ello desea consultar el valor de las acciones de esta, además de otros datos financieros. Sin embargo, estas consultas al no ser anónimas, pueden advertir a la empresa pequeña de los posibles planes.

Con esta idea se empezó a investigar recientemente la forma de crear modelos de bases de datos y procedimientos que sean capaces de garantizar este nivel de seguridad a los usuarios. A estos métodos que fueron apareciendo se los denominó protocolos PIR, del inglés Private Information Retrieval (Recuperación Privada de la Información).

En el caso de una base de datos formada por un único servidor, el único protocolo PIR posible consiste en descargar la totalidad de la base de datos, cosa que queremos evitar ya que resulta muy costoso. Es por ello, que a la hora de desarrollar un protocolo PIR, inicialmente debemos desarrollar un modelo de base de datos en el que existan varios servidores. A partir de este modelo se explica una manera de realizar solicitudes a los diferentes servidores para recuperar cierto archivo que deseamos. Desde el punto de vista de los servidores de la base de datos, la información que se le está solicitando no guarda ninguna relación con ninguno de los archivos que contiene. Sin embargo, desde el punto de vista del usuario, al agrupar las respuestas a las solicitudes de cierta forma, éste será capaz de recuperar el archivo que deseaba de forma privada.

Pongamos un primer ejemplo con una base de datos formada por dos servidores. La base de datos estará formada por $m$ archivos, donde cada uno de ellos es un elemento de un cuerpo finito $\mathbb{F}_q$. Podemos representar dicha base de datos como un vector $x$ de longitud $m$ sobre

$\mathbb{F}_q$. Supondremos entonces que cada uno de los dos servidores tienen almacenados la totalidad de los datos, es decir, el vector $x$. El siguiente procedimiento nos permitirá recuperar el archivo que deseemos $x_f$.

En primer lugar, empezamos generando un vector aleatorio , $u \in \mathbb{F}_q^m$. A partir de este vector le haremos una solicitud al primero de los servidores. Le solicitaremos el producto escalar entre $u$ y los datos que él posee. Su respuesta será por tanto $r_1 = x \cdot u \in \mathbb{F}_q$. El tamaño de la solicitud es pequeño ya que está formada por el vector $u$ únicamente. Desde el punto de vista de este servidor, no estamos interesados en ninguno de los archivos en concreto, sino en una combinación de todos los datos, que no guarda ninguna relación con alguno de ellos particularmente. Al segundo servidor, en cambio, le solicitaremos el producto escalar entre sus datos y el vector $u + e_f$, donde $e_f$ se trata del vector canónico con un uno en la posición $f$. Desde el punto de vista de este servidor tampoco estamos interesados en ningún archivo en concreto. Su respuesta será $r_2 = x \cdot (u + e_f) \in \mathbb{F}_q$. Sin embargo, al restar las dos respuestas podremos obtener $x_f$, ya que $r_2 - r_1 = x \cdot (u + e_f) - x \cdot u = x \cdot e_f = x_f$.

El principal problema de este protocolo es que si ambos servidores se comunican entre ellos, al comparar las dos solicitudes que han recibido, podrían llegar a deducir, usando el mismo procedimiento que hemos hecho nosotros, que estamos interesados en el archivo $x_f$. A estos dos servidores que son capaces de comunicarse entre ellos los llamamos servidores cooperantes y presentan un grave problema en la elaboración de estos protocolos. En este ejemplo en concreto, nuestro protocolo solo funciona cuando la cantidad de servidores cooperante es a lo sumo $b = 1$.

Si para este mismo ejemplo $b = 2$ la única forma válida de obtener el archivo $x_f$ de forma privada sería descargarse la base de datos al completo, ya que cualquier conclusión que podamos sacar nosotros a partir de las respuestas, también podrían obtenerla los propios servidores. Por la misma razón, en cualquier base de datos que esté formada por un único servidor, esta será la única solución.

Es por ello que estos protocolos solamente funcionan en bases de datos formadas por varios servidores, las llamadas bases de datos distribuidas. De hecho, este tipo de base de datos es la más frecuente hoy en día ya que facilita el acceso a datos en cualquier parte del mundo.

En este ejemplo que hemos visto se usa una base de datos en la que toda la información está almacenada en cada uno de los servidores. Se conoce como una base de datos replicada. Este tipo de bases tiene la ventaja de que contactando con cualquier servidor podremos acceder a la totalidad de la información. No obstante, el principal problema de este tipo, es que si el tamaño de dicha base es muy elevado, replicarla en muchos servidores puede ser muy costoso.

Por ello, se empezó a usar un segundo tipo de bases de datos distribuidas. En este nuevo tipo, cada servidor únicamente contiene parte de la información total, así que en todos los casos, para obtener la totalidad de un archivo deberemos contactar con varios servidores. Para garantizar ciertas propiedades, la réplica de la información no está hecha de forma aleatoria, sino usando un código lineal. Cada archivo se divide en varios fragmentos que son posteriormente codificados usando un código lineal. Gracias a ello, el usuario podrá recuperar la información que desee contactando con cierto número de servidores cualesquiera. Además el código lineal nos dará también la opción de corregir errores y borrones que puedan surgir en la transmisión de la información.

Los protocolos PIR sobre este caso concreto de base de datos serán el principal objeto de

estudio en este documento. En él, nos centraremos en explicar detalladamente, analizar e implementar los protocolos descritos en cuatro artículos:

- R. Tajeddine, O.W. Gnilke, S. El Roayheb *Private Information Retrieval From MDS Coded Data in Distributed Storage Systems*, IEEE Transactions on Information Theory 64 (2018) 7081-7093 [5]
- R. Freij-Hollanti, O. Gnilke, C. Hollanti, D. Karpuk, *Private Information Retrieval from Coded Databases with Colluding Servers*, SIAM J. Appl. Algebra Geometry 1 ( 2017), 647–664 [7]
- R. Tajeddine, O.W. Gnilke, D. Karpuk, R. Freij-Hollanti, C. Hollanti, *Robust Private Information Retrieval from Coded Systems with Byzantine and Colluding Servers*, arXiv 1802.03731 [10]
- R. Tajeddine, O.W. Gnilke, D. Karpuk, R. Freij-Hollanti, , *Private Information Retrieval from Coded Storage Systems with Colluding, Byzantine, and Unresponsive Servers*, IEEE Transactions on Information Theory 65 (2019) 3898 - 3906 [11]

En cuanto a la organización de este documento, se hará de la siguiente forma.

En el primer capítulo empezaremos con la teoría que nos permitirá desarrollar los protocolos más adelante. En primer lugar veremos conceptos básicos de teoría de la información como la entropía y la información mutua. Estos dos conceptos nos permitirán definir en términos matemáticos el concepto de "privacidad". También gracias a ellos seremos capaces de probar matemáticamente en qué casos los servidores pueden deducir o no información a partir de las solicitudes que los enviamos.

En ese mismo capítulo también indagaremos en la teoría de códigos lineales, ya que estos serán fundamentales para la construcción de los protocolos PIR. Además de usarlos para codificar bases de datos, también construiremos códigos a partir de los cuales generamos las solicitudes a los servidores. También introduciremos unos códigos que poseen propiedades que resultan muy interesantes para usarlos en la codificación de la base de datos. Estos códigos son los códigos Reed-Solomon y Reed-Solomon Generalizados. Al ser códigos MDS resultan muy útiles en la maximización de la distancia mínima para usarlos en la codificación de los archivos en la base de datos. También tienen la propiedad de funcionar excepcionalmente bien cuando se realiza el producto estrella entre ellos. Este producto estrella es una operación entre códigos que veremos más adelante. La dimensión del producto entre códigos arbitrarios crece en gran medida, sin embargo, entre códigos GRS este crecimiento está controlado.

Una vez que hemos introducido los conceptos básicos en el capítulo anterior, empezaremos el capítulo 2 con los protocolos PIR. Inicialmente explicaremos el modelo principal que usaremos a lo largo del documento. En él veremos cómo está formada la base de datos a partir de un código lineal y cuál es el proceso para realizar las solicitudes a los servidores. Una vez explicados estos conceptos, nos centraremos en los protocolos introducidos en el primero de los artículos [5]. Veremos tres protocolos PIR, todos ellos útiles para el caso en el que la base de datos ha sido codificada usando un código MDS con matriz sistemática. El primero de ellos protegerá únicamente contra un servidor cooperante. El segundo, en cambio, protegerá hasta contra $d-1$ de ellos, donde $d$ es la distancia mínima del código usado para codificar la base de datos. El tercero de ellos será una mejora del anterior para

casos en los que el número de servidores cooperantes es pequeño en comparación con la longitud del código.

En el capítulo 3 empezaremos introduciendo una nueva operación sobre códigos y vectores, el producto estrella. Esta nueva operación nos permitirá desarrollar nuevos protocolos para tipos diferentes de códigos. En concreto, lo utilizaremos para adaptar los protocolos del capítulo anterior en el caso de un código arbitrario. El resultado de esta adaptación es el protocolo explicado en [7]. Esta versión tendrá la ventaja de que, como hemos dicho, el código usado para codificar la base de datos puede ser un código arbitrario. La principal innovación que se realiza en este protocolo es la de usar un segundo código $D$ para generar las solicitudes, de forma que las respuestas pertenezcan al producto estrella de $D$ y el código de la base de datos $C$. Sin embargo, dependiendo de la elección que hagamos de $D$, seremos capaces de protegernos contra una mayor cantidad de servidores cooperantes. Además del caso general, también nos fijaremos en el caso en el que $C$ es un código GRS. En este caso, veremos que la elección óptima del código $D$ se trata de otro código GRS de mismos parámetros que el inicial pero de diferente dimensión.

Por último, en el cuarto capítulo, realizaremos un estudio de los dos protocolos descritos en los dos últimos artículos [10],[11]. Ambos se centran en el caso en el que la información se transmite en un canal con ruido y por tanto es posible recibir errores y/o borrones en las respuestas de los servidores. El primero de los protocolos será una versión inicial en la que además de un segundo código $D$ para generar las solicitudes, emplearemos un tercero $E$ de dimensión 1 que nos permitirá corregir los errores y borrones antes de comenzar a recuperar los fragmentos del archivo. En el segundo nos centraremos en el caso que $C$ es un código GRS y adaptaremos el anterior, usando un código $E$ de mayor dimensión para lograr una mejora en la eficiencia.

Además de la explicación de los protocolos, en todos los casos se han acompañado de ejemplos para facilitar su comprensión. En algunos casos, los ejemplos han sido realizados en *Sage* para realizar de forma más rápida los cálculos, ya que a veces pueden ser complejos.

Al final del documento, se incluye un anexo con la implementación de todos los protocolos explicados en el trabajo de fin de grado y algunos otros métodos que pueden resultar útiles en la simulación de estos protocolos en *Sage*. Todos los programas comienzan con un comentario de ayuda en el que se explica la utilidad de cada programa y una descripción de los parámetros de entrada. Si se desea una explicación más detallada de estos programas, esta se encuentra en el TFG de informática.

## Objetivos

A continuación presentamos los objetivos principales que se desean alcanzar durante la realización de este documento:

- Estudiar la teoría básica de códigos lineales
- Conocer los códigos Reed-Solomon y Reed-Solomon Generalizados, tanto su construcción como sus principales algoritmos de codificación y decodificación.
- Aprender los conceptos básicos de teoría de información necesarios para comprender el concepto de “privacidad” en los protocolos PIR.

- Aprender los conceptos básicos del lenguaje de programación Python
- Conocer la funcionalidad que ofrece el lenguaje de programación *Sage* [12] en cuanto a códigos lineales, códigos Reed-Solomon y otras funciones básicas del mismo.
- Implementar en *Sage* métodos básicos para la codificación de archivos en bases de datos usando códigos lineales.
- Leer los cuatro artículos [5],[7],[10],[11] donde se describen los principales protocolos PIR para bases de datos distribuidas codificadas usando un código lineal. Para cada uno de estos protocolos se realizará una descripción detallada de la base de datos, se explicará la construcción de las solicitudes y del análisis de las respuestas. Por último se demostrará su funcionamiento.
- Analizar la eficiencia de cada uno de estos protocolos y estudiar la protección que se realiza contra la presencia de servidores cooperantes.
- Realizar la implementación en *Sage* de cada uno de estos protocolos PIR.

# Capítulo 1

# Teoría de la información y Códigos Correctores Lineales

En este capítulo explicaremos conceptos fundamentales de dos campos que usaremos a menudo a lo largo del documento. La primera de ellas será la teoría de la información, y más adelante veremos teoría de códigos lineales.

## 1.1. Teoría de la información

La teoría de la información es la rama de las matemáticas que estudia la transmisión y el procesamiento de información. Entre otras cosas una de las tareas de las que se encarga es la medición y representación de ésta. Para ello introduciremos el concepto de entropía con el que se puede medir la incertidumbre de una variable aleatoria. Gran parte de este capítulo esta basado en [6].

**Definición 1.1.** Sea $X$ una variable aleatoria discreta sobre un alfabeto $\mathcal{X}$. Definimos la entropía de $X$ como:

$$H(X) = -\sum_{x \in \mathcal{X}} p(x) \log p(x)$$

Si el logaritmo es de base 2 entonces diremos que la entropía está expresada en bits. Si en cambio el logaritmo es en base $e$ diremos que está expresado en *nats*. Podemos expresarla en cualquier base $q$. En ese caso lo escribiremos $H_q(X)$ y la mediremos en q-bits. Generalmente se suele usar la entropía en base 2.

**Lema 1.2.** *Sea $X$ una variable aleatoria, se tiene entonces que $H(X) \geq 0$*

*Demostración.* $p(x) \geq 0$ para todo $x$ y $\log p(x) \leq 0$. Por tanto cada sumando es negativo y al cambiar de signo el total nos queda positivo. □

**Lema 1.3.** *Sea $X$ una variable aleatoria, entonces $H(X) = 0$ si y solo si $X$ toma un único valor con probabilidad 1.*

*Demostración.* Si $X$ toma un único valor, entonces $H(X) = p(x) \log p(x) = 1 \log 1 = 0$.

Si $X$ toma al menos dos valores, entonces para cada uno de los valores de $x$, $p(x) \neq 0$ y $\log p(x) \neq 0$. Como todos los sumandos tienen signo negativo la suma es distinta de 0. $\square$

También podemos definir la entropía para la distribución conjunta de varias variables aleatorias discretas en vez de una única de la siguiente forma

**Definición 1.4.** Sean $X, Y$ un par de variables aleatorias discretas con una función de distribución conjunta $p(x, y)$. Definimos la entropía conjunta de $X, Y$ de la siguiente forma:

$$H(X,Y) = -\sum_{x\in\mathcal{X}}\sum_{y\in\mathcal{Y}} p(x,y)\log p(x,y)$$

De hecho es fácil ver que $H(X, Y) = H(Y, X)$ ya que podemos intercambiar el orden de los sumatorios.

**Definición 1.5.** Sean $X, Y$ dos variables aleatorias. La entropía condicional $H(Y|X)$ se define como:

$$H(Y|X) = \sum_{x\in\mathcal{X}} p(x)H(Y|X=x)$$

Podemos desarrollar la definición para obtener otra forma de expresarla:

$$\begin{aligned} H(Y|X) &= \sum_{x\in\mathcal{X}} p(x)H(Y|X=x) = -\sum_{x\in\mathcal{X}} p(x)\sum_{y\in\mathcal{Y}} p(y|x)\log p(y|x) \\ &= -\sum_{x\in\mathcal{X}}\sum_{y\in\mathcal{Y}} p(x)p(y|x)\log p(y|x) = -\sum_{x\in\mathcal{X}}\sum_{y\in\mathcal{Y}} p(x,y)\log p(y|x) \end{aligned}$$

Veremos ahora como están relacionadas las dos formas de entropía que acabamos de definir

**Proposición 1.6.**

$$H(X,Y) = H(X) + H(X|Y)$$

*Demostración.*

$$\begin{aligned} H(X,Y) &= -\sum_{x\in\mathcal{X}}\sum_{y\in\mathcal{Y}} p(x,y)\log p(x,y) \\ &= -\sum_{x\in\mathcal{X}}\sum_{y\in\mathcal{Y}} p(x,y)\log p(x)p(y|x) \\ &= -\sum_{x\in\mathcal{X}}\sum_{y\in\mathcal{Y}} p(x,y)\log p(x) - \sum_{x\in\mathcal{X}}\sum_{y\in\mathcal{Y}} p(x,y)\log p(y|x) \\ &= -\sum_{x\in\mathcal{X}} p(x)\log p(x) - \sum_{x\in\mathcal{X}}\sum_{y\in\mathcal{Y}} p(x,y)\log p(y|x) \\ &= H(X) + H(Y|X) \end{aligned}$$

$\square$

**Ejemplo 1.7.** Sean $X_1$, $X_2$ dos variables aleatorias que simulan el lanzamiento de una moneda, por lo que siguen una distribución de Bernouilli de parámetro $1/2$. Sean $X, Y$ las variables aleatorias $X = X_1 + X_2$, $Y = X_1 - X_2$. Podemos calcular las siguientes entropías:

$$H(X) = H(Y) = -\sum_{x\in\mathcal{X}} p(x)\log p(x) = -\frac{1}{2}\log_2\left(\frac{1}{2}\right) - \frac{1}{4}\log_2\left(\frac{1}{4}\right) - \frac{1}{4}\log_2\left(\frac{1}{4}\right) = \frac{3}{2}$$

$$H(X|Y) = \frac{1}{4}H(X|_{Y=-1}) + \frac{1}{4}H(X|_{Y=1}) + \frac{1}{2}H(X|_{Y=0}) = \frac{1}{2}H(X|_{Y=0}) = \frac{1}{2}$$
$$H(X,Y) = H(X|Y) + H(X) = 2$$

**Definición 1.8.** Sean $X, Y$ dos variables aleatorias con probabilidad conjunta $p(x,y)$ y funciones de probabilidad marginales $p(x), p(y)$. Definimos la información mutua como la entropía relativa entre la distribución conjunta y el producto de las marginales:

$$I(X,Y) = \sum_{x\in\mathcal{X}}\sum_{y\in\mathcal{Y}} p(x,y)\log\frac{p(x,y)}{p(x)p(y)}$$

De la propia definición se puede deducir que es un operador simétrico.

Este nuevo concepto se puede relacionar también con las entropías de las variables $X, Y$

**Proposición 1.9.**
$$I(X,Y) = H(X) - H(X|Y)$$

*Demostración.*

$$\begin{aligned}
I(X,Y) &= \sum_{x\in\mathcal{X}}\sum_{y\in\mathcal{Y}} p(x,y)\log\frac{p(x,y)}{p(x)p(y)} \\
&= \sum_{x\in\mathcal{X}}\sum_{y\in\mathcal{Y}} p(x,y)\log\frac{p(y)p(x|y)}{p(x)p(y)} \\
&= \sum_{x\in\mathcal{X}}\sum_{y\in\mathcal{Y}} p(x,y)\log\frac{p(x|y)}{p(x)} \\
&= -\sum_{x\in\mathcal{X}}\sum_{y\in\mathcal{Y}} p(x,y)\log p(x) + \sum_{x\in\mathcal{X}}\sum_{y\in\mathcal{Y}} p(x,y)\log p(x|y) \\
&= -\sum_{x\in\mathcal{X}} p(x)\log p(x) + \sum_{x\in\mathcal{X}}\sum_{y\in\mathcal{Y}} p(x,y)\log p(x|y) \\
&= H(X) - H(X|Y)
\end{aligned}$$

□

**Definición 1.10.** También definimos la información mutua condicional de las variables aleatorias $X, Y$ con respecto a $Z$ como:

$$I(X,Y|Z) = H(X|Z) - H(X|Y,Z)$$

**Ejemplo 1.11.** Tomamos las variables del ejemplo 1.7 para calcular la información mutua.

$$I(X,Y) = H(X) - H(X|Y) = 1$$

## 1.2. Códigos lineales

A la hora de transmitir datos a través de un canal nos encontramos con un problema, la alteración de información. Cuando se manda un mensaje codificado, por ejemplo en una onda, interferencias con otras ondas pueden cambiar la forma de esta, haciendo que en el proceso de decodificación el mensaje obtenido sea diferente al enviado. Una solución para este problema es añadir información redundante al mensaje, de forma que si se pierde algún elemento del mensaje, éste se pueda determinar a partir del resto.

Consideramos por tanto un mensaje $m$ que se codifica a otro mensaje $c$. Los alfabetos de la palabra mensaje y la palabra codificada no tienen por qué coincidir. Sin embargo consideraremos únicamente el caso en el que ambos alfabetos son iguales, los mensajes sin codificar son de longitud $k$ y los codificados son de longitud $n$. Estos códigos se denominan códigos en bloque de tipo [n,k]. Trabajaremos exclusivamente con los denominados códigos lineales.

**Definición 1.12.** Sea $\mathbb{F}_q$ un cuerpo finito y sea $\mathbb{F}_q^n$ el conjunto de n-uplas sobre $\mathbb{F}_q$. Llamamos un *código lineal* $C$ de longitud $n$ a un subespacio vectorial no vacío de $\mathbb{F}_q^n$. Llamaremos a los elementos de $C$ *palabras código*. La dimensión de un código lineal $k$ se define como su dimensión como subespacio vectorial. Denotamos este código por sus parámetros $[n,k]_q$ o simplemente $[n,k]$. Definimos además la redundancia del código como $n-k$.

**Definición 1.13.** Sea $C$ un código lineal de tipo [$n$,$k$]. Un codificador de $C$ es una aplicación inyectiva

$$\mathcal{E} : \mathbb{F}_q^k \to \mathbb{F}_q^n$$

tal que $C = \mathcal{E}(\mathbb{F}_q^k)$. De esta forma para cada palabra código $c$ en $\mathbb{F}_q^n$ existe una única *palabra fuente* $m \in \mathbb{F}_q^k$ tal que $c = \mathcal{E}(m)$.

Ahora nos interesa el estudio de la capacidad correctora de nuestro código. Para ello nos resulta útil definir una distancia sobre las diferentes palabras código. La métrica que se usa en estos casos es la métrica de Hamming.

**Definición 1.14.** Para $x = (x_1, ..., x_n)$, $y = (y_1, ..., y_n) \in \mathbb{F}_q^n$ la distancia de Hamming entre $x$ e $y$, $d(x,y)$ se define como el número de posiciones en las que ambos difieren:

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

Se puede comprobar de forma sencilla que la distancia de Hamming cumple las tres condiciones para ser considerada una métrica.

Ahora que tenemos una métrica sobre las palabras de nuestro código puede resultar interesante estudiar la distancia entre ellas.

**Definición 1.15.** La distancia mínima de un código lineal $C \subseteq \mathbb{F}_q^n$ se define como

$$d = d(C) = min\{d(x,y) | x, y \in C, x \neq y\}$$

si $C$ contiene más de una palabra. Entonces decimos que el código $C$ tiene parámetros $[n,k,d]$.

**Ejemplo 1.16.** El código de repetición de longitud $n$ sobre un cuerpo $\mathbb{F}_q$ consiste en el conjunto de n-uplas en $\mathbb{F}_q^n$ que tienen todas las coordenadas iguales. Se trata de un código de dimensión 1 y longitud $n$. Tiene distancia mínima $n$. Un codificador de este código es la aplicación $\mathcal{E} : x \to (x, ..., x)$.

A continuación definiremos dos nuevos operadores sobre las palabras código que nos ayudarán a dar una definición alternativa de la distancia mínima de un código. Esta nueva forma será mucho más simple de calcular en la práctica.

**Definición 1.17.** Para una palabra c $\in \mathbb{F}_q^n$ definimos el *soporte* sop(c) como el conjunto de coordenadas distintas de cero: $sop(c) = \{i | c_i \neq 0\}$. Definimos también su *peso* $w(c)$ como el número de elementos en su soporte. Definimos el peso mínimo de un código $w(C)$ como el mínimo del peso de todos sus elementos.

**Proposición 1.18.** *La distancia mínima $d$ de un código es igual a su peso mínimo*

*Demostración.* En primer lugar dado que $0 \in C$ siempre y $w(c) = d(0, c)$ para cada vector $c \in C$, tenemos que $d \leq w(C)$. Sean ahora $c_1$, $c_2 \in C$ tal que $d = d(c_1, c_2)$. Entonces $d = d(c_1, c_2) = d(0, c_2 - c_1) = w(c_2 - c_1)$ y como $c_1 - c_2 \in C$ tenemos la desigualdad en la otra dirección. □

**Ejemplo 1.19.** Definiremos un código binario (sobre $\mathbb{F}_2$) de dimensión 4 y longitud 7 de la siguiente forma. Una palabra $(x_1, x_2, x_3, x_4)$ se codifica añadiéndola 3 bits de redundancia $(r_1, r_2, r_3)$. De forma que

$$r_1 = x_2 + x_3 + x_4$$

$$r_2 = x_1 + x_3 + x_4$$

$$r_2 = x_1 + x_2 + x_4$$

Este código se denomina código de Hamming(7,4). Para calcular su distancia mínima buscaremos una palabra de peso mínimo. En primer lugar, la palabra de peso 3, $(1, 0, 0, 0, 0, 1, 1)$ pertenece al código y por tanto $d \leq 3$.

Veremos ahora que no hay ninguna palabra de peso 2. Si las cuatro coordenadas iniciales son 0, la redundancia también. Si entre las cuatro primeras coordenadas, únicamente una de ellas, $x_i$, es distinta de 0, entonces para los $j \neq i$, $r_j = 1$ y habrá al menos dos coordenadas distintas de 0. Por último, en una palabra con dos coordenadas no nulas, $x_i, x_j$, al menos uno de los bits de redundancia $r_k$ con $k \neq i, j$ será distinto de cero y por tanto tendrá también al menos peso 3. Por tanto el código de Hamming(7,4) tiene distancia mínima 3.

## 1.3. Matriz Generadora y Codificación Sistemática

Hemos visto que un código lineal es un subespacio vectorial. Por tanto podemos aplicar los conceptos de álgebra lineal para definir explícitamente el código.

Sea $C$ un código lineal de tipo $[n, k]$ sobre $\mathbb{F}_q$. Dado que $C$ es de dimensión $k$, existen $k$ vectores linealmente independientes $g_1, ..., g_k$ que forman una base del subespacio. Si $g_i = (g_{i1}, ..., g_{in})$ entonces podemos definir la matriz:

$$G = \begin{pmatrix} g_{11} & g_{12} & \cdots & g_{1n} \\ g_{21} & g_{22} & \cdots & g_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ g_{k1} & g_{k4} & \cdots & g_{kn} \end{pmatrix}$$

Cada palabra código puede ser escrita como una única combinación lineal de los elementos de esta base. Por tanto para cada $m \in \mathbb{F}_q^k$, $mG \in C$. De la misma forma para toda palabra $c \in C$ existe un $m \in \mathbb{F}_q^k$ tal que $mG = c$.

Podemos usar esta matriz $G$ para realizar la codificación $\mathcal{E} : \mathbb{F}_q^k \to \mathbb{F}_q^n$ de forma eficiente.

**Definición 1.20.** Una matriz $k \times n$ con entradas en $\mathbb{F}_q$ se denomina una matriz generatriz de un código lineal si sus filas forman una base de $C$.

Un mismo código puede tener varias matrices generatrices distintas pero todas deben de tener rango $k$. Igualmente toda matriz $k \times n$ de rango $k$ genera un código [n,k].

**Ejemplo 1.21.** El código de Hamming antes descrito en el ejemplo 1.19 es claro que está generado por la codificación de los 4 vectores canónicos de $\mathbb{F}_2^4$ y por tanto tendrá matriz generatriz

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

Ahora daremos una serie de resultados de álgebra lineal que nos serán de ayuda para definiciones posteriores.

**Proposición 1.22.** *Sea $G$ una matriz generatriz de $C$. Entonces la matriz escalonada equivalente por filas a $G$ genera también $C$.*

*Demostración.* La matriz escalonada equivalente por filas se obtiene realizando operaciones elementales por filas a G. Estas operaciones por filas no cambian el subespacio generado por la matriz. □

**Proposición 1.23.** *Sean $G_1, G_2$ dos matrices generatrices $k \times n$ de dos códigos $C_1, C_2$ sobre $\mathbb{F}_q$. Es equivalente entonces:*

1. $C_1 = C_2$
2. *$G_1$ y $G_2$ son equivalentes por filas a la misma matriz escalonada*
3. *Existe una matriz invertible $M$ tal que $G_1 = MG_2$*

*Demostración.* Los resultados son consecuencia de resultados elementales de álgebra lineal. □

A la hora de codificar una palabra, como nuestro objetivo es el de añadir información para evitar su pérdida o modificación en el proceso de transmisión, podría ser interesante que la propia palabra a codificar sea parte de la palabra código, incluso sea el principio de la misma. Introduciremos ahora nuevos conceptos para dar una definición matemática a esta idea.

**Definición 1.24.** Sea $C$ un código de tipo [n,k]. El código se denomina *sistemático* en las posiciones $(j_1, ..., j_k)$ si para todo $m \in \mathbb{F}_q^k$ existe una única palabra código $c$ tal que $c_{j_i} = m_i$.

Una matriz generatriz G de $C$ es *sistemática* en las posiciones $(j_1, ..., j_k)$ si la submatriz $k \times k$ formada por las columnas $(j_1, ..., j_k)$ es la identidad. Para este tipo de matriz decimos que la codificación es sistemática.

Si $j_i = i$ para todo $i \leq k$ podemos decir en ambos casos simplemente que es *sistemático.*

**Proposición 1.25.** *Sea C un código con matriz generatriz G. Entonces C es sistemático en las posiciones* $(j_1, ..., j_k)$ *si y solo si las k columnas de G en dichas posiciones son linealmente independientes.*

*Demostración.* En primer lugar, si $C$ es sistemático en las posiciones $(j_1, ..., j_k)$, la matriz formada por esas columnas $G'$ genera una aplicación inyectiva y por tanto son linealmente independientes.

Para la otra dirección si suponemos que las columnas $(j_1, ..., j_k)$ son linealmente independientes existirá una matriz invertible tal que $MG'$ es la identidad. Por tanto la matriz $MG$ generará el mismo código $C$ y éste es sistemático en las posiciones $(j_1, ..., j_k)$. □

**Definición 1.26.** Llamamos a dos códigos $C_1$ y $C_2$ *equivalentes* si existe una permutación $\sigma$ de n elementos tal que $C_2 = \{\sigma(c) | c \in C_1\}$.

**Proposición 1.27.** *Todo código lineal es equivalente a un código sistemático. Además la matriz generatriz de este código se puede conseguir sistemática.*

*Demostración.* Sea $G$ una matriz generatriz de $C$. Entonces $G$ es de rango $k$ y por tanto posee $k$ columnas independientes. Por tanto podemos realizar una permutación $\sigma$ a las columnas de $G$ para que las $k$ columnas independientes pasen a ser las primeras. De esta forma obtendremos una matriz $G' = (A|B)$ donde A es invertible. Ahora podemos realizar operaciones elementales sobre las filas de $G'$ para obtener su matriz escalonada equivalente por filas. Como las primeras $k$ columnas eran linealmente independientes, la submatriz formada por las $k$ primeras columnas será la identidad. □

## 1.4. Matriz de Control y Código Dual

De la misma forma que podemos definir un subespacio vectorial a partir de sus generadores, también lo podemos definir implícitamente usando un sistema de ecuaciones lineales. Usaremos este concepto para definir de una nueva forma un código y veremos cómo se relaciona con la otra forma que teníamos de definirlo.

**Definición 1.28.** Una matriz de tamaño $(n-k)\times n$ de rango máximo es una matriz de control de un código $C$ de tipo $[n,k]$ si para todo $c\in\mathbb{F}_q^n$ se cumple que $Hc^T=0$ si y solo si $c\in C$.

**Proposición 1.29.** *Sea $C$ un código de tipo $[n,k]$. Sea $G$ su matriz generatriz y sea $H$ una matriz de tamaño $(n-k)\times n$ de rango máximo. Entonces $H$ es una matriz de control de $C$ si y solo si $GH^T=0$.*

*Demostración.* Sea $H$ una matriz de control. Entonces para todo $x\in\mathbb{F}_q^k$, $xG\in C$. Entonces $HG^Tx^T=0$ lo que implica que $xGH^T=0$ para todo $x\in\mathbb{F}_q^k$. Por tanto $GH^T=0$.

Para la otra dirección tendremos en cuenta que $H$ es la matriz de control para algún código $C'$ de tipo $[n,k]$. Para cualquier elemento $c\in C$ tenemos que $c=xG$. Entonces $Hc^T=HG^Tx^T=0$ y por tanto $H$ es la matriz de control de $C$ ya que $C\subseteq C'$ y ambos son de la misma dimensión y por tanto iguales. □

**Proposición 1.30.** *Sea $C$ un código de tipo $[n,k]$. Sea $I_k$ la identidad de tamaño $k$ y sea $P$ una matriz de tamaño $k\times(n-k)$. Entonces $(I_k|P)$ es una matriz generatriz de $C$ si y solo si $(-P^T|I_{n-k})$ es una matriz de control de $C$.*

*Demostración.* Sea $H$ una matriz de control de un código $C$. $G$ es matriz generatriz de cierto código $C'$. Dado que $GH^T=-P+P=0$ podemos usar la proposición anterior para probar que entonces $C=C'$. □

Ahora veremos una forma útil de obtener la distancia mínima de un código a partir de su matriz de control.

**Proposición 1.31.** *Sea $H$ una matriz de control de cierto código $C$. Entonces la distancia mínima $d=d(C)$ es el menor entero $d$ tal que existen $d$ columnas linealmente dependientes.*

*Demostración.* Sea $c\neq 0\in C$. Sea $(j_1,...,j_l)$ su soporte. Entonces $Hc^T=0$ y por tanto para cada $j_i$, $c_{j_1}h_{j_1}+...+c_{j_l}h_{j_l}=0$ con $c_{j_i}\neq 0$. Por tanto las columnas $(j_1,...,j_l)$ son linealmente dependientes.

De la otra forma si las columnas $(j_1,...,j_l)$ son linealmente dependientes, existe un vector c tal que $Hc^T=0$. Si elegimos c de forma que $c_j=0$ si $j\neq j_i$ entonces $w(c)\leq l$. Además como $H$ es matriz de control, $c\in C$. □

Dado que tanto la matriz de control como la matriz generatriz son matrices de rango máximo que definen un código, tiene sentido plantearse que código es el resultante de usar la matriz de control como una generatriz y viceversa. El resultado es el llamado código dual.

**Ejemplo 1.32.** Para el código de Hamming descrito anteriormente en el ejemplo 1.19, la matriz de control es la siguiente:

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

Esta matriz viene dada por las ecuaciones implícitas que nos definían originalmente como se obtenían los bits de redundancia del código. Podemos observar en este ejemplo cómo se cumplen todos los resultados de las proposiciones anteriores.

**Definición 1.33.** Para un código $C$ de tipo $[n, k]$ definimos su código dual $C^{\perp}$:

$$C^{\perp} = \{x \in \mathbb{F}_q^n | c \cdot x = 0, \forall c \in C\}$$

**Proposición 1.34.** *Sea $C$ un código de tipo $[n, k]$ con matriz generatriz $G$. Entonces $C^{\perp}$ es un código de tipo $[n, n-k]$ con matriz de control $G$.*

*Demostración.* Sea $x \in C^{\perp}$. Entonces $c \cdot x = 0$ para todo $c \in C$. Por tanto $mG^T x^T = 0$ para todo $m \in \mathbb{F}_q^k$ y por consiguiente $Gx^T = 0$. $C^{\perp}$ está contenido en el núcleo de la matriz. De la misma forma un elemento del núcleo de $G$ tiene que estar en $C$ ya que cumple que $mGx^T = 0$ para todo $m \in \mathbb{F}_q^k$. Como $G$ tiene rango $k$, $C^{\perp}$ es de dimensión $n-k$ y $G$ es su matriz de control. □

**Proposición 1.35.** *Sea $C$ un código lineal de tipo $[n, k]$. Se tiene entonces*

1. $(C^{\perp})^{\perp} = C$
2. $C^{\perp} = C \Leftrightarrow x \cdot y = 0 \ \forall x, y \in C$ *y además* $n = 2k$.

*En este último caso entonces se denomina a $C$ un código autodual.*

*Demostración.* (1) Sea $c \in C$. Entonces $c \cdot x = 0$ para todo $x \in C^{\perp}$. Por tanto $C \subseteq (C \perp)^{\perp}$. Aplicando la proposición anterior dos veces obtenemos que la dimensión de ambos es igual y por tanto son iguales.

(2)Si la primera condición se cumple entonces $C \subseteq C^{\perp}$. La igualdad solo se dará si la dimensión de ambos es igual, es decir si $k = n - k$. Lo que es lo mismo que $n = 2k$. □

## 1.5. Decodificación

Ahora que sabemos las diferentes formas representar un código y cómo codificarlo estudiaremos el posterior. Supondremos que hemos recibido una palabra ya codificada. Tendremos en cuenta que la palabra que hemos recibido puede tener errores, borrones o incluso ambas y trataremos de decodificarla corrigiendo los posibles errores que han ocurrido durante la transmisión.

**Definición 1.36.** Sea $C$ un código lineal sobre $\mathbb{F}_q^n$ con distancia mínima $d$. Si se ha transmitido una palabra código $c$ y se ha recibido una palabra $r$, entonces el conjunto de posiciones erróneas es $\{i | r_i \neq c_i\}$. El cardinal de este conjunto es el número de errores cometidos. Llamamos el vector de error a $e = r - c$.

Una vez que hemos recibido una palabra $r$, existe una palabra $c'$ que es la palabra del código más próxima a $r$. Lo lógico es decodificar $r$ por esa palabra $c'$ suponiendo que se han cometido la menor cantidad de errores posible. Pero no siempre podemos garantizar que esta palabra sea única. Sin embargo existe un caso en el que sí. Si sabemos que se

han cometido a lo sumo $t = \lfloor (d-1)/2 \rfloor$ errores, entonces esta palabra $c'$ es única y podemos hacer la decodificación correctamente. En el caso contrario podemos afirmar que ha ocurrido un error en la transmisión pero seremos incapaz de corregirlo. Llamaremos a $t$ la capacidad correctora del código.

Una forma intuitiva de ver este resultado es en la figura 1.1. En ella están representados los distintos puntos de un código y bolas abiertas de radio d/2 alrededor de ellos. Como el radio es $d/2$ estas son disjuntas. Si en una palabra recibida se han cometido menos de $d/2$ errores entonces esta estará en el interior de la bola y por tanto existirá una única palabra del código más cercana a ella. En el caso contrario (por ejemplo el punto donde se cortan las dos circunferencias) no podemos garantizar que haya una única palabra código más cercana.

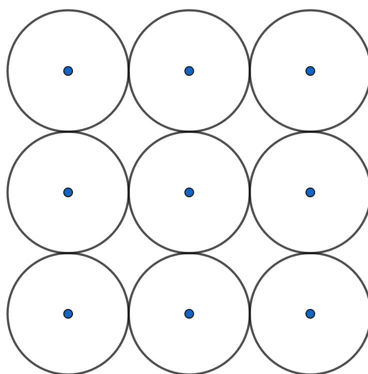

Figura 1.1: Bolas de radio $d/2$ alrededor de las palabras de un código

Además de corregir errores también podemos corregir borrones, es decir, posiciones de la palabra que no se pueden identificar con ningún símbolo del abecedario. En este caso podremos corregir $d-1$ errores simplemente resolviendo un sistema lineal $Hr^T = 0$ donde los borrones en $r$ son las incógnitas. Un error que conocemos en qué posición se halla podemos tratarlo al igual que un borrón.

En el caso de haber $b$ borrones y $e$ errores podremos corregirlos siempre que $2e+b \leq d-1$. Para hacerlo proyectamos el código en las $n-b$ coordenadas donde no hay borrones. De esta forma obtendremos un código de tipo $[n-b, k, d-b]$. La proyección de la palabra que queremos decodificar ahora tiene únicamente $e$ errores. Como $2e \leq d-b-1$ entonces podremos corregir estos errores como habíamos visto. Ahora con los errores ya corregidos podremos hacer lo mismo con los borrones que faltan para terminar la decodificación de la palabra.

**Ejemplo 1.37.** Consideraremos el código de repetición sobre $\mathbb{F}_3$ de longitud 5. Supongamos que la palabra recibida es $r = (2, 1, \bullet, 2, \bullet)$ donde $\bullet$ denota un borrón. Dado que hay dos borrones la decodificación será correcta si hay como máximo un único error.

En primer lugar proyectamos el código sobre las coordenadas 1,2 y 4. De esta forma obtenemos un código de repetición de longitud 3. La palabra proyectada será $r' = (2, 1, 2)$. La palabra código más cercana a ésta es $c = (2, 2, 2)$, por tanto corregimos la posición errónea y obtenemos entonces la palabra $(2, 2, \bullet, 2, \bullet)$. Ahora es momento de corregir los

borrones. La matriz de control del código de repetición es

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

Tenemos que resolver $Hs^T = 0$ para $s = (2, 2, x_1, 2, x_2)$. Obtenemos entonces $x_1 = 2$ y $x_2 = 2$ como era esperado y por tanto la palabra enviada es $(2, 2, 2, 2, 2)$.

Introduciremos ahora un método para calcular esta palabra más cercana $c'$.

**Definición 1.38.** Sea $C$ un código de dimensión $k$ y sea $H$ una matriz de control para $C$. Sea $r$ una palabra recibida. Entonces $s = Hr^T$ es el *síndrome de $r$ respecto a $H$.*

Consideramos la relación de equivalencia que relaciona dos palabras $x_1$, $x_2$ si y solo si $x_1 - x_2 \in C$. Entonces dos elementos serán de la misma clase de equivalencia si y solo si tienen el mismo síndrome ya que $r_1 - r_2 \in C \Leftrightarrow H(r_1 - r_2)^T = 0 \Leftrightarrow Hr_1^T = Hr_2^T$. Como en cada clase hay $|C| = q^k$ elementos, entonces existen por tanto $q^{n-k}$ distintas clases de equivalencia y por tanto la misma cantidad de síndromes.

**Definición 1.39.** Si en una clase de equivalencia existe un único elemento de peso mínimo, denominamos a este elemento el líder de la clase de equivalencia.

**Proposición 1.40.** *Todas las clases de equivalencia tienen como máximo un único elemento de peso $\leq t$.*

*Demostración.* Sean $u$, $v$ dos elementos de peso $\leq t$ en la misma clase de equivalencia, entonces $u - v \in C$. $w(u - v) \leq w(u) + w(v) \leq 2t \leq d$. Por tanto $u - v = 0$ y $u = v$. □

Podemos usar el líder de una clase de equivalencia para decodificar una palabra recibida $r$ o en otras palabras, encontrar la única palabra $c \in C$ más cercana a $r$. $c$ es la palabra en $C$ que hace mínimo a $w(r - x)$, $x \in C$. Como para todo $x \in C$, $r - x$ está en la misma clase de equivalencia que $r$ entonces $r - c$ será el líder de la clase $e$ (en caso de que exista) y por tanto $c = r - e$. La proposición anterior nos garantiza además que si el número de errores es menor que la capacidad correctora del código, entonces la decodificación es correcta.

El único problema es calcular el líder de una clase a partir del síndrome. Para solucionar este problema se construye previamente a la decodificación una tabla en la que se relaciona cada síndrome con su líder (si existe). Esta tabla se usa en el algoritmo descrito para no tener que computar el líder cada vez que se quiera decodificar una palabra.

**Algoritmo 1.41.** *Se recibe un vector $r$.*

1. *Se calcula $s(r)$*

2. *Se busca en la tabla de síndromes el líder correspondiente. Si no existe, la decodificación falla*

3. *Si la clase tiene líder $e$, la palabra decodificada es $r - e$*

**Ejemplo 1.42.** Consideramos el código de Hamming del ejemplo 1.19. Este código tenía distancia mínima 3 así que corrige 1 error. La tabla de síndromes es la siguiente:

| Síndrome | Líder | Síndrome | Líder |
|---|---|---|---|
| 000 | 0000000 | 100 | 0000100 |
| 001 | 0000001 | 101 | 0100000 |
| 010 | 0000001 | 110 | 0010000 |
| 011 | 1000000 | 111 | 0001000 |

Usaremos esto para corregir los errores. Supongamos que recibimos $r = (1, 0, 0, 1, 0, 0, 1)$. Lo primero que haremos será calcular su síndrome $s = Hr^T = (1, 0, 1)$. Ahora vamos a la tabla de síndromes y vemos que el líder existe y es $(0, 1, 0, 0, 0, 0, 0)$. Por tanto obtenemos que el vector emitido es $c = r - e = (1, 1, 0, 1, 0, 0, 1)$.

## 1.6. Códigos MDS

**Teorema 1.43.** *(Cota de Singleton) Sea $C$ un código de tipo $[n, k, d]$. Entonces*

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

*Demostración.* En $C$ hay un total de $q^k$ palabras distintas. Como la distancia mínima entre dos palabras es $d$ si eliminamos las primeras $d - 1$ componentes de cada palabra estas seguirán siendo diferentes. Estas nuevas palabras tendrán $n - d + 1$ componentes. Esto quiere decir que la proyección de las $q^k$ palabras de $C$ está contenida en $\mathbb{F}_q^{n-d+1}$ y por tanto $k \leq n - d + 1$. Reordenando esta última ecuación obtenemos la desigualdad deseada. □

Está demostración es válida incluso considerando códigos no lineales. Esto da una idea de que un código lineal que satisfaga la igualad en la cota será muy bueno incluso teniendo en cuenta los códigos no lineales.

Por tanto la mayor distancia que puede tener un código de tipo $[n, k]$ viene dada por la anterior desigualdad. Tiene sentido entonces considerar si la igualdad se puede alcanzar y en ese caso buscar códigos que la cumplan. Veremos que estos códigos existen y son los llamados códigos MDS y veremos también cuáles son.

**Definición 1.44.** Un código lineal de tipo $[n, k, d]$ es de distancia máxima separable (MDS) si $d = n - k + 1$.

**Definición 1.45.** (Códigos Reed-Solomon) Sea $x_1, ..., x_n$ elementos distintos de un cuerpo finito $\mathbb{F}_q$. Para $k \leq n$ consideramos el conjunto $\mathbb{P}_k$ de polinomios sobre $\mathbb{F}_q$ de grado menor que $k$. Un código de Reed-Solomon consiste de las palabras

$$\{(f(x_1), ..., f(x_n)), \ \ para\ f \in \mathbb{P}_k\}$$

Este código es claramente lineal por la linealidad de los polinomios. Los polinomios $\mathbb{P}_k$ forman un espacio vectorial sobre $\mathbb{F}_q$. Además la aplicación que lleva un polinomio al elemento del código es inyectiva ya que si un polinomio de grado menor que $k$ se anula en más de $k$ puntos entonces es 0. Por eso el código es de dimensión $k$. Por tanto es claro que se trata de un código lineal de tipo $[n, k]$.

**Ejemplo 1.46.** Sea $\mathbb{F}_7$ el cuerpo de 7 elementos. Tomamos $x_1 = 0$, $x_2 = 1$, ..., $x_7 = 6$. Calcularemos un código Reed-Solomon sistemático de tipo $[7, 3, 5]$ sobre $\mathbb{F}_7$. Como el código será de dimensión 3 estará generado por 3 palabras de longitud 7 $v_1, v_2, v_3$ que al ser sistemático podemos suponer que las 3 primeras posiciones de estas palabras se corresponden con los vectores canónicos de $\mathbb{F}_7^3$. Para cada una de estas palabras $i$ buscaremos ahora el único polinomio $p_i$ en $\mathbb{P}_3$ tal que $p_i(x_j) = v_j$ para $j \leq 3$. Evaluando este polinomio en los 7 puntos obtendremos las palabras que generan el código. Obtenemos lo siguiente:

$$p_1 = 4(x-1)(x-2) \to p_1(x) = (1, 0, 0, 1, 3, 6, 3)$$

$$p_2 = 6x(x-2) \to p_2(x) = (0, 1, 0, 4, 6, 6, 4)$$

$$p_3 = 2x(x-1) \to p_3(x) = (0, 0, 1, 3, 6, 3, 1)$$

Todas las palabras del código serán generadas por estos 3 vectores. Por tanto formaran la matriz generatriz:

$$G = \begin{pmatrix} 1 & 0 & 0 & 1 & 3 & 6 & 3 \\ 0 & 1 & 0 & 4 & 6 & 6 & 4 \\ 0 & 0 & 1 & 3 & 6 & 3 & 1 \end{pmatrix}$$

Para codificar una palabra $(a, b, c)$ tomamos el polinomio $ax^2+bx+c$ y evaluamos los 7 puntos obteniendo una nueva palabra. Por ejemplo la palabra $(5, 3, 6)$ generaría el polinomio $5x^2+3x+6$. Si lo evaluamos en los 7 puntos obtenemos la palabra código $(6, 0, 4, 4, 0, 6, 1)$.

**Proposición 1.47.** *Un código Reed-Solomon es MDS.*

*Demostración.* Buscamos una palabra del código de peso mínimo. Una palabra del código puede tener como máximo $k-1$ ceros en el caso de que $k-1$ del los $x_1, ..., x_n$ sean raíces de $f$. Esto se debe a que $f$ es de grado menor que $k$ y por el teorema fundamental del álgebra tiene como máximo $k-1$ raíces. Además sabemos que existe un polinomio $f'$ que tiene a $x_1, ..., x_{k-1}$ como raíces. Existe por tanto un polinomio que alcanza el peso menor posible y este peso es $n-(k-1)$. Por tanto la distancia del código será la misma, $n-k+1$. □

La codificación se puede realizar evaluando el polinomio en cada uno de los puntos. Buscaremos ahora una forma eficiente de decodificar las palabras código. La idea para la decodificación de una palabra $r = c+e$ tal que $w(e) \leq t$ donde t es la capacidad correctora del código, es determinar un polinomio en dos variables

$$Q(x, y) = Q_0(x) + yQ_1(x) \in \mathbb{F}_q[x, y] \setminus \{0\}$$

que cumpla

1. $Q(x_i, r_i) = 0,\ i = 1, ..., n$
2. $deg(Q_0) \leq n-1-t$
3. $deg(Q_1) \leq n-1-t-(k-1)$

**Teorema 1.48.** *Existe un polinomio $Q(x, y)$ que cumple las condiciones dadas anteriormente.*

*Demostración.* Un polinomio que cumpla las condiciones 2 y 3 estará formado por $(n-1-t+1)+(n-1-t-k+1+1) = 2n-2t-k+1 \geq n+1$ coeficientes distintos. Como la primera condición añade n ecuaciones lineales homogéneas sobre estos al menos $n+1$ coeficientes, existirá una solución distinta de cero. □

**Teorema 1.49.** *Si la palabra transmitida está generada por el polinomio $g(x)$ y el número de errores es menor que $\frac{d}{2}$ entonces $g(x) = \frac{-Q_0(x)}{Q_1(x)}$.*

*Demostración.* Sea $c = (g(x_1), ..., g(x_n))$ y $r = c + e$ con $wt(e) \leq t$. Por construcción $Q(x_i, g(x_i) + e_i) = 0$ para cada i. Además como $e_i = 0$ para al menos $n-t$ posiciones, tenemos que entonces el polinomio $Q(x, g(x))$ tiene al menos $n-t$ ceros que son los $x_i$ donde no ha ocurrido un error en la transmisión. Pero el grado de este polinomio es $n-1-t-(k-1)+k-1 = n-1-t$. Por tanto $Q(x, g(x)) = 0$. Entonces $Q_0(x)+g(x)Q_1(x) = 0 \Leftrightarrow g(x) = -\frac{Q_0(x)}{Q_1(x)}$. □

Podemos entonces ahora usar esto para desarrollar el siguiente algoritmo para corregir los errores y decodificar una palabra del código. Para simplificar $I_0 = n-1-t$, $I_1 = n-1-t-(k-1)$.

**Algoritmo 1.50.** *Entrada:* $r = (r_1, ..., r_n)$

- *Resolvemos el sistema de ecuaciones correspondiente a la primera condición*

$$\begin{bmatrix} 1 & x_1 & x_1^2 & ... & x_1^{I_0} & r_1 & r_1x_1 & ... & r_1x_1^{I_1} \\ 1 & x_2 & x_2^2 & ... & x_2^{I_0} & r_2 & r_2x_2 & ... & r_2x_2^{I_1} \\ \vdots & \vdots & \vdots & ... & \vdots & \vdots & \vdots & ... & \vdots \\ 1 & x_n & x_n^2 & ... & x_n^{I_0} & r_n & r_nx_n & ... & r_nx_n^{I_1} \end{bmatrix} \begin{bmatrix} Q_{0,0} \\ Q_{0,1} \\ Q_{0,2} \\ \vdots \\ Q_{0,I_0} \\ Q_{1,0} \\ Q_{1,1} \\ Q_{1,2} \\ \vdots \\ Q_{1,I_1} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 0 \\ 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

- *Construimos los polinomios*

$$Q_0(x) = \sum_{j=0}^{I_0} Q_{0,j}x^j \quad Q_0(x) = \sum_{j=0}^{I_1} Q_{1,j}x^j \quad g(x) = -\frac{Q_0(x)}{Q_1(x)}$$

- *Si $g(x) \in \mathbb{P}_k$ entonces devolvemos $(g(x_1), g(x_2), ..., g(x_n))$*

  *En caso contrario falla.*

La complejidad de este algoritmo consiste en la resolución de un sistema lineal, es decir $O(n^3)$ en el caso de la eliminación gaussiana, que es más simple que calcular los líderes de los $q^{n-k}$ síndromes distintos. Especialmente cuando n es algo grande ya que q lo es aún más.

Podemos realizar una pequeña variación en la definición del código anterior para definir una versión generalizada del código. Este nuevo código que definiremos también cumplirá la cota.

**Ejemplo 1.51.** Consideramos el código Reed-Solomon del ejemplo 1.46. Supongamos que hemos recibido la palabra $r = (3, 2, 6, 3, 4, 2, 1)$ y queremos decodificarla. Supondremos que se han cometido a lo sumo dos errores. En primer lugar tenemos que calcular los coeficientes de los polinomios $Q_0(x), Q_1(x)$ que serán de grado $n - 1 - t = 4$ y $n - 1 - t - (k - 1) = 2$ respectivamente. Para ello resolveremos el siguiente sistema lineal de ecuaciones.

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 3 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 2 & 2 & 2 \\ 1 & 2 & 4 & 1 & 2 & 6 & 5 & 3 \\ 1 & 3 & 2 & 6 & 4 & 3 & 2 & 6 \\ 1 & 4 & 2 & 1 & 4 & 4 & 2 & 1 \\ 1 & 5 & 4 & 6 & 2 & 2 & 3 & 1 \\ 1 & 6 & 1 & 6 & 1 & 1 & 6 & 1 \end{pmatrix} \begin{pmatrix} Q_{0,0} \\ Q_{0,1} \\ Q_{0,2} \\ Q_{0,3} \\ Q_{0,4} \\ Q_{1,0} \\ Q_{1,1} \\ Q_{1,2} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

Los polinomios resultantes de resolver el sistema son: $Q_1(x) = 3 + x + x6x^2 + 6x^4 + 5x^4$, $Q_2(x) = 6 + x^2$ por lo que el polinomio $g(x) = -Q_0(x)/Q_1(x)$ es $g(x) = 3 + x + 2x^2$.

Ahora solo falta calcular el valor de $g$ en cada uno de los $x_i$. Obtenemos la palabra $(3, 6, 6, 3, 4, 2, 4)$ por lo que se habían cometido 2 errores.

**Definición 1.52.** (Códigos Reed-Solomon Generalizados) Sea $\mathbb{F}_q$ el cuerpo finito de $q$ elementos. Elegimos $n$ elementos distintos del cuerpo $x = (x_1, ..., x_n)$ al igual que para los códigos Reed-Solomon. Consideramos igualmente los polinomios en $\mathbb{P}_k$. Sea $v = (v_1, ..., v_n)$ un vector de elementos de $\mathbb{F}_q$ no nulos. Definimos entonces el código Reed-Solomon Generalizado como el conjunto de las palabras

$$\{(v_1 f(x_1), v_2 f(x_2), ..., v_n f(x_n)), \ \ para\ f \in \mathbb{P}_k\}$$

Este código podemos escribirlo como $GRS_k(x, v)$.

Además la matriz generatriz $G$ de un código Reed-Solomon Generalizado es de la forma

$$G = \begin{pmatrix} v_1 & v_2 & ... & v_n \\ v_1 x_1 & v_2 x_2 & ... & v_n x_n \\ v_1 x_1^2 & v_2 x_2^2 & ... & v_n x_n^2 \\ \vdots & \vdots & \ddots & \vdots \\ v_1 x_1^{k-1} & v_2 x_2^{k-1} & ... & v_n x_n^{k-1} \end{pmatrix} = \begin{pmatrix} 1 & 1 & ... & 1 \\ x_1 & x_2 & ... & x_n \\ x_1^2 & x_2^2 & ... & x_n^2 \\ \vdots & \vdots & \ddots & \vdots \\ x_1^{k-1} & x_2^{k-1} & ... & x_n^{k-1} \end{pmatrix} \cdot diag(v)$$

**Proposición 1.53.** *Los códigos Reed-Solomon Generalizados son MDS.*

*Demostración.* La demostración es idéntica al caso de los códigos Reed-Solomon. Dado que los $v_i$ son distintos de 0 podemos aplicar el teorema fundamental del álgebra de la misma forma. □

Veremos ahora un resultado sobre el dual de un código de este tipo que nos ayudará a definir un nuevo código MDS.

**Proposición 1.54.** *El código dual de un código Reed-Solomon Generalizado es también Reed-Solomon Generalizado.*

*Demostración.* Sea $C$ un código Reed-Solomon Generalizado de tipo $[n, k, n-k+1]$ generado por el vector $v$ y los elementos del cuerpo $x_1, .., x_n$. Nos vale probar que la matriz de control de un código Reed-Solomon Generalizado es de la misma forma que en la definición 1.52. Sea $H$ la matriz generatriz de un código Reed-Solomon Generalizado de tipo $[n, n-k, k+1]$ generada por un vector $w$ y los mismos $x_i$ que en $G$. Probaremos que podemos elegir $w$ tal que $GH^T = 0$. Tenemos que imponer que el producto de cada fila de $G$ por cada fila de $H$ es 0, es decir, $\sum_{i=1}^{n} v_i x_i^a w_i x_i^b = v_i w_i x_i^{a+b} = 0$ para todos los $a, b$. Por tanto nos vale imponerlo para cada $0 \leq a+b \leq n-2$ ya que $j \leq k-1$ y $k \leq n-k-1$. Pero esto es un sistema de $n-1$ ecuaciones lineales homogéneas para las $n$ incógnitas que tenemos y por tanto el sistema tiene solución. Podemos elegir por tanto una matriz de control de $C$ que genera un código Reed-Solomon Generalizado. □

Un código Reed-Solomon generalizado $C$ es por tanto un código MDS. Queremos ahora extender este código para obtener otro de una longitud mayor $\check{C}$ que también sea MDS. Para ello definimos la matriz generatriz de un código $\check{C}$, $\check{G} = [G, u_1^T]$ donde $u_1 = (0, ..., 0, a)$ para un $0 \neq a \in \mathbb{F}_q$, y $G$, la matriz generatriz de $C$. Hemos visto que el código dual de $C$ está generado por un vector $w$ y los mismos $x_i$ y esta definido para polinomios de grado menor que $n-k$. Definimos ahora $H$ como la matriz generatriz de un código Reed-Solomon Generalizado con estos mismos generadores pero para polinomios de un grado mayor, es decir, de grado menor o igual que $n-k$. Podemos comprobar entonces que para cierto $b$, la matriz de control de $\check{C}$ es $\check{H} = [H|u_2^T]$ donde $u_2 = (0, ..., 0, b)$. Elegiremos este $b$ de forma que el producto de las dos últimas filas de $\check{G}$ y $\check{H}$ sea 0.

**Ejemplo 1.55.** Usaremos el código del ejemplo 1.46 para crear un código extendido como hemos explicado. La matriz generatriz de este código será la siguiente:

$$\check{G} = \begin{pmatrix} 1 & 0 & 0 & 1 & 3 & 6 & 3 & 0 \\ 0 & 1 & 0 & 4 & 6 & 6 & 4 & 0 \\ 0 & 0 & 1 & 3 & 6 & 3 & 1 & 1 \end{pmatrix}$$

La utilidad de extender un código Reed-Solomon de esta forma es que ,como veremos en la siguiente proposición, el código resultante sigue siendo un código MDS.

**Proposición 1.56.** *El código extendido $\check{C}$ es MDS.*

*Demostración.* Tomamos $n-k+1$ columnas de $\check{H}$ para una submatriz cuadrada. Si ninguna de las columnas que hemos tomado es la última, entonces podemos descomponer la submatriz $M$ en el producto de una matriz diagonal formada por los $w_i$ correspondientes a las columnas que hemos elegido y otra matriz de Vandermonde. $M$ es por tanto invertible y las columnas son linealmente independientes. En el caso de que una de las columnas seleccionadas sea la última nos fijaremos en las primeras $n-k$ coordenadas. Estas son las mismas que las de la matriz de control de $C$ y como $C$ es MDS son linealmente independientes. Esto quiere decir que si ya nos fijamos en la fila al completo ninguna combinación lineal de ellas podrá tener como resultado la última fila ya que las primeras $n-k$ coordenadas son 0. Por tanto son linealmente independientes. Usando la proposición 1.31 tenemos entonces que la distancia mínima es al menos n-k+2. Usando la cota de Singleton obtenemos la igualdad y por tanto es MDS. □

Ahora que sabemos cómo es el dual de un código Reed-Solomon Generalizado también nos podemos plantear si se da el mismo caso para los códigos MDS en general.

**Proposición 1.57.** *El dual de un código MDS es también MDS*

*Demostración.* Sea $C$ MDS. Supongamos que su dual no es MDS. Entonces existe en $C^{\perp}$ una palabra $v$ de peso $w(v) \leq k$. Podemos ampliar v hasta una base de $C^{\perp}$ y con ellos construir una matriz generatriz en la que cada fila es un elemento de la base. Esta matriz sería también una matriz de control de $C$. Por la proposición 1.31 cualquier $n-k$ columnas de $H$ son linealmente independientes. Tomamos entonces las $n-k$ que tienen un 0 en la primera coordenada. Estos $n-k$ vectores de dimensión $n-k$ son linealmente independientes y tienen todos un 0 en la primera coordenada, lo cual es imposible. □

Por último veremos una conjetura que de ser cierta caracterizaría bastante a este tipo de códigos.

**Conjetura 1.58.** (Conjetura MDS) Sea $C$ un código MDS sobre $\mathbb{F}_q$ de tipo $[n, k]$. Entonces $n \leq q+1$ excepto si $q$ es par y $k = 3$ ó $k = q-1$. En este caso $n \leq q+2$.

# Capítulo 2

# Modelo PIR y primeros protocolos

Acceder a una base de datos es algo que hacemos muchas veces a diario. Este proceso en principio es bastante simple. El usuario envía una solicitud y la base de datos devuelve la información que el usuario solicita. Hoy en día existen muchos métodos de proteger la privacidad en una base de datos para que, por ejemplo, un usuario sólo pueda acceder a una parte de ella. Sin embargo no se pone tanto empeño en proteger la privacidad del propio usuario. En la solicitud que envía un usuario a un servidor, está especificado qué parte de la base de datos quiere obtener el usuario, y el propio servidor puede almacenar un historial de todas nuestras solicitudes sin que nosotros podamos evitarlo. Sin embargo existen protocolos para obtener la información de servidores sin que éstos sepan en que parte de los datos que tiene almacenado estamos interesados. Estos protocolos de obtención de información son los llamados PIR, del inglés Private Information Retrieval. Ejemplos de situaciones en la que éstos métodos podrían ser usados es la de una persona que no quiera ser espiada por un régimen opresor o un inversor que desea datos sobre el mercado de valores pero no desea que se sepa en que acciones está interesado. Un primer protocolo de estas características sería el de solicitar la base de datos entera, pero como se supone que ésta es muy grande, no resulta práctico. Buscaremos por tanto protocolos en el que la cantidad de datos solicitada a la base sea lo menor posible.

En ésta época es normal que los datos no estén almacenados únicamente en un servidor. Este tipo de almacenamiento podría ser peligroso para una organización ya que en caso de un fallo del servidor, toda la información dejaría de estar disponible. Existen varias soluciones pero todas ellas se basan en replicar la información en diferentes servidores. Una de estas formas sería replicar toda la información en varios servidores. Así en el caso de que uno de ellos falle, el resto seguirán disponibles para las distintas solicitudes de los usuarios.

Podemos suponer por tanto para un primer ejemplo, que la base de datos está formada por $n$ servidores. Cada uno de ellos almacena todos los datos. Estos datos además están formados por una serie de $m$ valores de $\mathbb{F}_q$, $(x_1 \dots x_m) = x$. El usuario está interesado en el valor $x_f$ de un índice $f$. Para obtener este valor, el usuario enviará una solicitud $Q_j \in \mathbb{F}_q^m$ a cada servidor $j$. La respuesta del servidor será el producto escalar de ambos vectores $r_j = Q_j x$.

Una primera forma de conseguir este elemento sin que cada servidor sepa que componente de $x$ estamos solicitando es la siguiente. Suponemos que estamos interesados en la com-

ponente $f$ de $x$. Generamos un vector aleatorio $u \in \mathbb{F}_q^m$ y elegimos dos de los servidores $i, j$ a los que enviaremos una solicitud. Para el primero usamos el propio vector $u$ como solicitud. En el segundo enviaremos el vector $u + e_f$ donde $e_f$ es el vector canónico con un 1 en la componente $f$ y 0 en el resto. Cada servidor no puede saber en que valor estamos realmente interesados, ya que desde su punto de vista estamos solicitando una combinación aleatoria de los valores que tiene almacenados. Sin embargo nosotros podemos poner en común las dos respuestas y de esta forma obtenemos el valor deseado de la siguiente forma:

$$r_j - r_i = (u + e_f)x - ux = e_f x$$

Sin embargo, si los dos servidores trabajasen en conjunto, también podrían haber descubierto la componente en la que estamos interesados de la misma forma que hemos hecho nosotros. Decimos este protocolo entonces solo garantiza PIR para un único servidor cooperante. Daremos una definición de este concepto más adelante.

En este capítulo veremos protocolos para realizar este proceso en otro tipo de bases de datos, las bases de datos distribuidas. Veremos distintos protocolos considerando distintas cantidades de servidores cooperantes. Estos protocolos fueron originalmente introducidos en [5].

## 2.1. Modelo

Otra forma que existe de replicar los datos en $n$ distintos servidores es almacenar únicamente parte de los datos en cada servidor, de forma que para cierto número $l$, la información total se pueda obtener a partir de la información contenida en cualquier subconjunto de los servidores de tamaño $l$. Una forma de lograr esta propiedad es usando un código lineal. Podemos dividir la información en $k$ partes para un $k \leq n$ y usar un código lineal de tipo $[n, k]$ para obtener $n$ elementos del código distintos. Si almacenamos cada uno de ellos en uno de los servidores, estamos replicando los datos con esta propiedad. Este número de servidores que necesitamos para recuperar la información será $n - b$ donde $b$ es el número de borrones que puede corregir nuestro código, ya que podemos interpretar la información faltante como un borrón. Como hemos visto, la cantidad de borrones que un código puede corregir es $d - 1$ donde $d$ es la distancia mínima del código. Así $l = n - d + 1$. Es lógico intentar hacer $l$ lo más pequeño posible para un número fijo de servidores $n$ y partes $k$ y la única forma de hacerlo es elegir un código con una distancia mínima lo más grande posible. Por ello los códigos MDS son los códigos que mejor nos vienen obteniendo un $l = k$. Este tipo de bases de datos se denominan bases de datos distribuidas.

Consideramos entonces el caso de una base de datos distribuida. Esta estará formada por $n$ servidores. Supondremos que la base de datos tiene $m$ archivos diferentes del mismo tamaño. Además supondremos que ésta usa un código MDS de tipo $[n, k, d]$ sobre $\mathbb{F}_q$ para almacenar los archivos. Podemos suponer que este código es sistemático por la proposición 1.27. De esta forma los archivos seguirán disponibles aunque $d - 1$ servidores no estén disponibles. Supondremos que cada archivo está dividido en $\alpha$ fragmentos distintos a los que llamaremos bandas. Cada banda también estará dividida en $k$ bloques distintos. Cada bloque podemos representarlo como un elemento de $\mathbb{F}_{q^\omega}$ para cierto $\omega$ suficientemente grande. Por tanto cada archivo puede ser representado como una matriz $X^i$de tamaño $\alpha \times k$ con entradas en $\mathbb{F}_{q^\omega}$. El propósito de dividir cada banda en $k$ bloques es para

posteriormente poder codificar cada banda en los $n$ servidores usando el código MDS. Por tanto un archivo queda codificado de la siguiente forma:

$$X^i = \begin{pmatrix} x^i_{11} & x^i_{12} & ... & x^i_{1k} \\ x^i_{21} & x^i_{22} & ... & x^i_{2k} \\ \vdots & \vdots & \ddots & \vdots \\ x^i_{\alpha 1} & x^i_{\alpha 2} & ... & x^i_{\alpha k} \end{pmatrix}$$

Usando la representación que hemos dado de cada archivo, si los agrupamos todos, podemos representarlos en una única matriz de tamaño $m\alpha \times k$ obteniendo así una representación en una matriz $\mathcal{X}$ de toda la información en la base de datos

$$\mathcal{X} = \begin{pmatrix} X^1 \\ X^2 \\ \vdots \\ X^m \end{pmatrix}$$

Cada banda luego se codifica con la ayuda de un código MDS sistemático sobre $\mathbb{F}_q$. Nos vale que el código sea sobre $\mathbb{F}_q$ y no sobre $\mathbb{F}_{q^\omega}$. Esto se debe a que cada elemento de $\mathbb{F}_{q^\omega}$ se puede representar por un polinomio de grado menor que $\omega$ con coeficientes en $\mathbb{F}_q$. De esta forma podemos codificar los $\omega$ coeficientes del polinomio usando un código sobre $\mathbb{F}_q$.

Llamaremos $\Lambda$ a la matriz generatriz de este código que por ser sistemático tendrá la identidad como submatriz formada por las $k$ primeras columnas. Los datos ya codificados $\mathcal{X}\Lambda$ son almacenados por columnas en los diferentes servidores. Así el servidor j-ésimo tendrá almacenado el vector sobre $\mathbb{F}_{q^\omega}$, $w_j = (x^1_{1j}, \dots, x^1_{\alpha j}, \dots, x^m_{1j}, \dots, x^m_{\alpha j})$ si $j \leq k$ o una combinación lineal de las $k$ primeras columnas dada por el código, en caso contrario.

**Ejemplo 2.1.** Trabajaremos sobre el cuerpo $\mathbb{F}_7$ en una base de datos formada por 7 servidores de forma que a partir de 4 cualesquiera podemos recuperar el resto. Para codificar la información en los 7 servidores se usa el código del ejemplo 1.46. Recordamos que este código tenía matriz generatriz

$$G = \begin{pmatrix} 1 & 0 & 0 & 1 & 3 & 6 & 3 \\ 0 & 1 & 0 & 4 & 6 & 6 & 4 \\ 0 & 0 & 1 & 3 & 6 & 3 & 1 \end{pmatrix}$$

Suponemos que un archivo está formado por una serie de símbolos que dividiremos como habíamos explicado en $\alpha k$ partes iguales formadas cada una de ellas por cierto $z$ elementos del cuerpo. Buscaremos un $\omega$ suficientemente grande como para representar cada una de ellas por un elemento de $\mathbb{F}_{q^\omega}$ Claramente para $\omega = z$ esto se cumple ya que podemos usar los $z$ elementos del cuerpo para definir un polinomio de grado menor que $\omega$ que representa un elemento de dicho cuerpo.

Ahora para obtener los datos que almacena cada servidor agrupamos los $m$ archivos en una matriz, la cual multiplicamos por la matriz generatriz del código $G$ para obtener una matriz de tamaño $\alpha \times 7$. Cada columna se corresponderá a los datos que contiene uno de los servidores. Así obtendremos que la información que almacena cada servidor es la siguiente:

| Servidor 1 | Servidor 2 | Servidor 3 | Servidor 4 | $\dots$ | Servidor 7 |
|---|---|---|---|---|---|
| $x_{11}^1$ | $x_{12}^1$ | $x_{13}^1$ | $x_{11}^1 + 4x_{12}^1 + 3x_{13}^1$ | $\dots$ | $3x_{11}^1 + 4x_{12}^1 + x_{13}^1$ |
| $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\ddots$ | $\vdots$ |
| $x_{\alpha 1}^1$ | $x_{\alpha 2}^1$ | $x_{\alpha 3}^1$ | $x_{\alpha 1}^1 + 4x_{\alpha 2}^1 + 3x_{\alpha 3}^1$ | $\dots$ | $3x_{\alpha 1}^1 + 4x_{\alpha 2}^1 + x_{\alpha 3}^1$ |
| $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ |
| $x_{11}^m$ | $x_{12}^m$ | $x_{13}^m$ | $x_{11}^m + 4x_{12}^m + 3x_{13}^m$ | $\dots$ | $3x_{11}^m + 4x_{12}^m + x_{13}^m$ |
| $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\ddots$ | $\vdots$ |
| $x_{\alpha 1}^m$ | $x_{\alpha 2}^m$ | $x_{\alpha 3}^m$ | $x_{\alpha 1}^m + 4x_{\alpha 2}^m + 3x_{\alpha 3}^m$ | $\dots$ | $3x_{\alpha 1}^m + 4x_{\alpha 2}^m + x_{\alpha 3}^m$ |

Supondremos entonces que el usuario desea obtener el archivo $X^f$ donde $f \in [m] = \{1, ..., m\}$. Para ello enviará solicitudes a los distintos servidores. Pero supondremos que entre los $n$ servidores, hay $b$ de ellos que están en contacto y pueden poner en común las solicitudes que han obtenido para descubrir el valor de $f$. Diremos que hay $b$ servidores cooperantes. El usuario no sabe cuáles son y por ello tendrá que evitar que cualesquiera $b$ servidores puedan descifrar $f$ a partir de su solicitud. Los protocolos que estudiaremos para realizar esta labor serán los protocolos PIR lineales.

**Definición 2.2.** Un protocolo PIR es lineal sobre $\mathbb{F}_q$ y de dimensión $\rho$ si está formado por las siguientes dos fases:

1. Fase de solicitud: El usuario envía una solicitud a un subconjunto de los servidores. Cada una de las solicitudes viene dada por una matriz $Q_l$ sobre $\mathbb{F}_q$ de tamaño $\rho \times m\alpha$.
2. Fase de descarga: Cada servidor que ha recibido una solicitud la responde enviando la proyección de sus datos sobre la matriz recibida: $R_l = Q_l w_l \in (\mathbb{F}_q)^\rho$.

Cada fila de las matrices $Q_l$ se puede entender como una sub-solicitud y por tanto se hacen $\rho$ sub-solicitudes a cada servidor. De la misma forma cada fila de la respuesta $R_l$ es la sub-respuesta a cada una de las sub-solicitudes.

Nos falta dar una definición matemática al concepto de que un subconjunto de $b$ servidores no sea capaz de descubrir el indice $f$ del archivo que queremos obtener.

**Definición 2.3.** Se dice que un protocolo consigue PIR teórico de la información para $b$ servidores cooperantes si y solo si se cumple $H(f|Q_j, j \in J) = H(f)$ para todos los conjuntos $J \subseteq [n]$, $|J| = b$. Donde $H$ es la función entropía.

Por último daremos una forma de medir el coste en cuanto a descarga de datos que tenemos que asumir para conseguir la recuperación privada.

**Definición 2.4.** El precio de la privacidad en la comunicación, del inglés cPoP, es el cociente de la cantidad de bits descargada por el usuario como respuesta a sus solicitudes, entre la cantidad de bits del archivo deseado. En muchos artículos también se denomina este parámetro como razón ratez se expresa como el inverso del valor definido.

Solamente tomamos como coste la descarga de la información y no el envío de las matrices de solicitud $Q_i$ ya que ésta en comparación con la respuesta del servidor es muy pequeña. En cada sub-solicitud enviamos al servidor $m$ elementos de $\mathbb{F}_q$. En cambio en cada sub-respuesta recibimos un elemento de $\mathbb{F}_{q^\omega}$. Generalmente $\omega$ será un número mucho mayor que $m$ ya que $\omega$ es un fragmento de un archivo que suponemos arbitrariamente grande.

El objetivo es por tanto desarrollar un protocolo PIR lineal que nos permita obtener el archivo que buscamos $f$, consiguiendo PIR teórico de la información para cierto $b$ y tratando de obtener el menor valor posible de cPoP. Dependiendo de la relación entre $d$ y $b$ daremos distintos protocolos para conseguirlo.

## 2.2. Protocolo PIR para b=1

En esta sección daremos un protocolo PIR para el caso mas sencillo, cuando no hay servidores cooperantes y por tanto $b = 1$. Este protocolo tendrá un cPoP de $\frac{1}{1-k/n}$. Para ayudar a la lectura del protocolo se añade la siguiente tabla con el significado de cada uno de los parámetros y elementos que se introducirán en esta sección.

| **Variable** | **Significado** |
|---|---|
| $X^i$ | Archivo $i$ sin codificar, podemos representarlo como una matriz $\alpha \times k$ |
| $\alpha$ | Número de bandas en las que está dividido cada archivo. |
| $C$ | Código MDS usado para codificar los archivos en los servidores |
| $k$<br>$n$ | Dimensión de $C$<br>Número de servidores y longitud de $C$ |
| $d$ | Distancia mínima de $C$, como suponemos $C$ MDS, $d = n - k + 1$ |
| $b$ | Número de servidores cooperantes |
| $G$ | Matriz Generatriz de $C$, en forma sistemática |
| $Y^i$ | Cada uno de los archivos codificados en los servidores, $Y^i = X^i G$ |
| $m$ | Número de archivos en la base de datos |
| $f$ | Índice del archivo deseado |
| $w_l$ | Vector columna de datos contenidos en un servidor $l$ |
| $\rho$ | Número de solicitudes necesarias para obtener $X^f$ |
| $\beta$ | Cociente de $\alpha$ entre $k$ |
| $r$ | Resto de $\alpha$ entre $k$ |
| $Q_l$ | Matriz solicitud para un servidor $l$. Cada fila es la sub-solicitud $q_{i,l}$ |
| $U$ | Matriz aleatoria de tamaño $rho \times m\alpha$. Cada fila i la denominamos $U_i$ |
| $E_{f,l}$ | Matriz de unos y ceros para el servidor $l$ para obtener el archivo $f$ |
| $r_{li}$ | Respuesta del servidor $l$ a la sub-solicitud $i$ |

Asumiremos todo lo descrito en la sección anterior. Además este protocolo será de dimensión $\rho = k$ y supondremos que los archivos están divididos en $\alpha = d - 1 = n - k$ bandas. Por tanto habrá tantas bandas como servidores con información redundante. Para empezar escribimos $\alpha = \beta k + r$ para dos enteros $\beta > 0$ y $0 \leq r < k$. De esta forma podemos dividir las $\alpha$ bandas de cada archivo en $\beta$ grupos de $k$ bandas con $r$ de ellas sobrantes.

Tenemos que determinar las matrices de solicitud $Q_l$ para cada servidor $1 \leq l \leq n$. Hemos dicho que el protocolo será de dimensión $\rho = k$ y por tanto la matriz solicitud que mandemos a cada servidor será de tamaño $\rho \times m\alpha$. Para empezar, generamos una matriz aleatoria de este tamaño $U = [u_{ij}]$ de elementos de $\mathbb{F}_q$. Con el uso de esta matriz conseguimos confundir al servidor ya que usarla como solicitud significaría pedir una combinación aleatoria de datos al servidor.

La solicitud que haremos a cada servidor será la suma de esta matriz $U$ y otra matriz

$E_{f,l}$ donde $E_{f,l}$ será una matriz que dependerá tanto del servidor al que le hacemos la solicitud $l$, como del archivo en el que estamos interesados $f$. De ella extraeremos parte de la información que realmente queremos. A continuación buscaremos cómo deben ser estas matrices exactamente para conseguir nuestro objetivo.

Llamemos $U_i$ a la i-ésima fila de $U$. Podemos ver esta fila como una de las sub-solicitudes que hacemos a los servidores. Si usamos $U_i$ como sub-solicitud en $k$ servidores distintos, obtendremos como sub-respuesta los valores de $U_i w_l^T$ para $k$ servidores distintos. Entonces podremos calcular el valor de $U_i w_l^T$ para los $n-k$ servidores restantes usando el código MDS. Por tanto podemos usar cada uno de estos $n-k$ servidores para obtener uno de los elementos que componen el archivo $X^f$ de la siguiente forma. Dado que para este servidor $s$ hemos calculado $U_i w_s^T$ a partir de las respuestas de otros servidores, podemos enviar a este servidor la sub-solicitud $U_i + e_j$ donde $e_j$ es el vector canónico con un 1 en la posición d. La sub-respuesta que recibiremos será por tanto $(U_i + e_j)w_s^T = U_i w_s^T + e_j w_s^T$. Sustrayendo $U_i w_s^T$ a esta sub-respuesta tendremos $e_j w_s^T$ que se corresponde con el valor de la j-ésima banda almacenado en el servidor $s$.

De esta forma obtendríamos con esta sub-solicitud uno de los valores en los que estamos interesados. Debemos crear las matrices $E_{f,l}$ a partir de filas representando sub-solicitudes que nos permitan realizar el proceso anterior. Una vez que hemos calculado el valor de la banda $j$ en el servidor $s$, no nos interesa volver a solicitarlo y por tanto en otras sub-solicitudes no estamos interesados en volver a calcular este mismo valor de esta forma. De este razonamiento entonces podemos deducir tres características que deben cumplir nuestras matrices $E_{f,l}$.

1. En cada fila y cada columna de cada matriz $E_{f,l}$ habrá como máximo un único 1. La restricción en las filas viene dada para poder calcular de forma privada uno de los valores como hemos explicado y la restricción en la columna es para no calcular un mismo valor varias veces.

2. En cada fila, únicamente $n-k$ matrices $E_{f,l}$ tienen un 1. De esta forma podemos calcular los valores de $U_i w_l^T$ para todos los servidores $l$ y calcular $n-k$ valores de forma privada.

3. Si $j$ es el índice de una de las bandas del archivo que queremos obtener $f$, entonces únicamente $k$ matrices $E_{f,l}$ tienen un 1 en la j-ésima columna. Esto se debe a que cada banda del archivo se corresponde a una columna de la solicitud. Además al menos necesitamos $k$ valores de esa banda para reconstruir la banda original. De la misma forma nos vale solo con $k$ valores y no necesitamos recuperar más.

Ahora daremos un conjunto de matrices $E_{f,l}$ para $l = 1 \dots k$ y comprobaremos que efectivamente cumplen estas 3 propiedades que necesitamos.

$$E_{f,1} = \begin{bmatrix} 0_{k\times(f-1)\alpha} & \begin{matrix} I_{r\times r} \\ 0_{(k-r)\times r} \end{matrix} & 0_{k\times\beta k} & 0_{k\times(m-f)\alpha} \end{bmatrix}$$

Para $2 \leq l \leq k$, $E_{f,l}$ se obtienen a partir de $E_{f,l-1}$ rotando todas las filas cíclicamente una posición hacia abajo. $E_{f,l} = 0$ para $l \geq n-r+1$. Por último las restantes las podemos dividir en $\beta$ grupos de $k$ elementos.

Para cada grupo $s$, con $s = 1 \dots \beta$, estarán las matrices con índice $sk + 1 \leq l \leq sk + k$. Todas las matrices de un mismo grupo $s$ serán iguales y éstas serán la siguiente:

$$E_{f,l} = \begin{bmatrix} 0_{k\times(f-1)\alpha+r+(s-1)k} & I_{k\times k} & 0_{k\times(\beta-s)k+(m-f)\alpha} \end{bmatrix}$$

**Proposición 2.5.** *Las matrices $E_{f,l}$ descritas anteriormente cumplen las 3 condiciones que buscábamos.*

*Demostración.*

1. Dado que están formadas por submatrices de ceros y una única submatriz unidad sólo pueden tener un único 1 en cada fila y columna.

2. En las matrices correspondientes a los $k$ primeros servidores hay exactamente $r$ unos en cada sub-solicitud, ya que al tener $k$ filas, las $k$ rotaciones de filas hacen que los $r$ unos estén una vez en cada fila. Para el resto de matrices descritas, todas tienen un uno en cada fila. Como había un total de $\beta$ grupos de $k$ elementos, todas ellas hacen un total de $\beta k + r = d - 1 = n - k$.

3. Tenemos que ver que para una banda del archivo fija, exactamente $k$ matrices tienen un 1 en la columna correspondiente. Las columnas que se corresponden con el archivo en el que estamos interesados, son las que se corresponden a los índices $(f-1)\alpha < j \leq (f+1)\alpha$. Para las $r$ primeras columnas, hay un 1 en cada una de las $k$ primeras matrices y en las $\beta k$ restantes no hay ninguno. Las $\beta k$ columnas restantes las podemos dividir en $\beta$ grupos de $k$ columnas al igual que hicimos con las matrices con índices $k + 1 \leq l \leq \beta k + k$. Para una columna en el grupo $s$ las únicas matrices con un 1 en esa columna son las matrices del grupo $s$ que son exactamente $k$ matrices. Por tanto la condición se cumple tanto para todas las bandas del archivo.

□

**Ejemplo 2.6.** Veremos ahora en un ejemplo esta primera fase de solicitud. Supondremos que estamos trabajando en el cuerpo $\mathbb{F}_7$ con el mismo modelo del ejemplo 2.1 y que deseamos obtener el archivo 1. Para este caso $\alpha = n - k = 4$ así que cada archivo está formado por 4 bandas. Recordamos que usaba un código con matriz

$$G = \begin{pmatrix} 1 & 0 & 0 & 1 & 3 & 6 & 3 \\ 0 & 1 & 0 & 4 & 6 & 6 & 4 \\ 0 & 0 & 1 & 3 & 6 & 3 & 1 \end{pmatrix}$$

Primero generamos la matriz aleatoria de tamaño $3 \times 4m$, $U = [u_{ij}]$. Calcularemos ahora las matrices $E_{f,l}$ teniendo en cuenta que en este caso nos queda $r = 1$. Para ahorrar espacio únicamente daremos las primeras 4 columnas ya que el resto es la parte correspondiente a los otros archivos y por tanto es 0.

$$E_{1,1} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \quad E_{1,2} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \quad E_{1,3} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}$$

$$E_{1,4} = E_{1,5} = E_{1,6} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad E_{1,7} = 0$$

La matriz de solicitud $Q_l$ que enviaremos a un servidor $l$ será la suma de la matriz aleatoria $U$ y $E_{1,l}$.

**Proposición 2.7.** *El protocolo descrito anteriormente permite decodificar todos los datos del archivo deseado $f$.*

*Demostración.* Para cada sub-solicitud $i$, diferenciamos dos tipos distintos de servidores. Por la segunda propiedad de las matrices $E_{f,l}$ sabemos que habrá $k$ servidores que habrán recibido como sub-solicitud el vector $U_i^T$ y los $n-k$ restantes habrán recibido $U_i^T + e_{j_l}$ para ciertos $j_l$ distintos. La respuesta de un servidor $l$ en el primer grupo será

$$r_{li} = U_i^T w_l$$

Cada banda almacenada es una palabra del código, por tanto, una combinación lineal de bandas también lo será. De hecho $u_i^T \cdot (w_1, ..., w_l)$ es una combinación lineal de todas las bandas de la base de datos y además $r_{li}$ es la l-ésima componente de de esta palabra. Por tanto con la respuesta de estos $k$ servidores del primer grupo podemos usar el código MDS para calcular $U_i^T w_l$ para todos los $l \leq n$.

Las sub-solicitudes del segundo grupo hemos visto que son de la forma $U_i^T + e_{j_l}$. Por tanto la respuesta del servidor será $(U_i^T + e_{j_l})w_l = U_i^T w_l + e_{j_l} w_l$. Como hemos podido calcular anteriormente el valor de $U_i^T w_l$ podemos calcular el valor de $e_{j_l} w_l$ sustrayendo este valor de la respuesta. De hecho el resultado que obtenemos es el valor de la banda $j_l$ almacenado en el servidor $l$. Hemos obtenido de forma privada uno de los valores en los que estamos interesados. Decimos que lo hemos obtenido de forma privada ya que el servidor del que hemos obtenido esta información no tiene forma de saber que es lo que queremos. Para él solo estamos interesados en una combinación aleatoria de sus valores almacenados.

De la primera propiedad de las matrices $E_{j,l}$ vemos que cada uno de estos valores que podemos obtener de forma privada, solo puede ser obtenido por una única sub-solicitud. Además por la tercera propiedad sabemos que obtendremos $k$ valores de cada banda de forma privada, pudiendo recuperar a partir de ellos la banda al completo (o únicamente los valores sistemáticos que son los que realmente nos interesan). Como podemos obtener con cada sub-solicitud $n - k = \alpha$ valores y hay un total de $k$ sub-solicitudes podremos obtener los $\alpha k$ valores necesarios para reconstruir el archivo al completo. □

**Ejemplo 2.8.** Por último realizaremos el proceso de decodificación del archivo 1 para el ejemplo 2.6. Cada solicitud es una matriz de 3 filas, por lo que esta formada por 3 sub-solicitudes y por tanto la respuesta del servidor $l$ será un vector $r_l = Q_l w_l \in \mathbb{F}_{7^\omega}^3$. Realizaremos únicamente la decodificación de los símbolos obtenidos por la primera sub-solicitud.

Sea $I_l = U_1^T w_l$ para $l \leq 3$ donde $U_1$ es la primera fila de $U$. Tenemos el siguiente sistema de ecuaciones lineales formado por las respuestas de los servidores:

$$I_1 + x_{11}^1 = r_{11}$$

$$I_2 = r_{21}$$

$$I_3 = r_{31}$$
$$I_1 + 4I_2 + 3I_3 + x_{21} + 4x_{22} + 3x_{23} = r_{41}$$
$$3I_1 + 6I_2 + 6I_3 + 3x_{21} + 6x_{22} + 6x_{23} = r_{51}$$
$$6I_1 + 6I_2 + 3I_3 + 6x_{21} + 6x_{22} + 3x_{23} = r_{61}$$
$$3I_1 + 4I_2 + I_3 = r_{71}$$

La segunda, tercera y última ecuación forman un sistema lineal que puede ser resuelto y de donde podemos obtener el valor de $I_1, I_3, I_3$. Al obtener estas valores podemos obtener $x_{11}$ de la primera ecuación y $x_{21}, x_{22}, x_{23}$ de la cuarta, quinta y sexta. De esta forma hemos conseguido calcular 4 fragmentos del archivo en está sub-solicitud. Los 8 restantes los obtendremos de la misma forma en las dos sub-solicitudes restantes.

En la siguiente tabla representamos los fragmentos del archivo que tiene cada servidor y cuales y en que sub-solicitud han sido obtenidos de forma privada

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| Banda 1 | 1 | 2 | 3 | | | | |
| Banda 2 | | | | 1 | 1 | 1 | |
| Banda 3 | | | | 2 | 2 | 2 | |
| Banda 4 | | | | 3 | 3 | 3 | |

**Teorema 2.9.** *El protocolo descrito en esta sección garantiza PIR teórico de la información para $b = 1$ servidores cooperantes con un cPoP igual a*

$$cPoP = \frac{1}{1 - \frac{k}{n}}$$

En primer lugar tenemos que probar que garantiza PIR teórico de la información. Tenemos que ver que $H(f) = H(f|Q_l)$ para cada $l$. Nos valdrá con ver que $f$ es independiente de $Q_l$ ya que entonces ambas distribuciones serían la misma y por tanto tendrían la misma entropía. Dada una matriz $Q_l$, para todos los archivos $f$ existe una única matriz $U = Q_l - E_{j,l}$ tal que $Q_l$ sería la matriz enviada como solicitud al servidor $l$ por el protocolo. Como la distribución de la matriz $U$ es uniforme, todas estas posibles matrices tienen la misma probabilidad y por tanto $Q_l$ no nos añade información sobre $f$ por lo que ambas son independientes.

Por último lugar calcularemos el cPoP. Queremos obtener $\alpha k = (n-k)k$ valores y para obtenerlos hacemos $k$ sub-solicitudes a $n$ servidores. De esta forma el número de valores que obtenemos es $kn$. Lo cual hace un cPoP de

$$\frac{nk}{nk - k^2} = \frac{1}{1 - \frac{k^2}{nk}} = \frac{1}{1 - \frac{k}{n}}$$

## 2.3. Protocolo PIR para d-1 servidores cooperantes

Ahora consideraremos el caso en el que hay servidores cooperantes, como máximo $d-1$. Describiremos el protocolo para $b = d-1$. En el caso que $b < d-1$ mandaremos solicitud

únicamente a los $b+k$ primeros servidores. De esta forma estaríamos actuando como si los datos estuvieran codificados usando un código como el original pero en el que se ignoran las últimas columnas. Este nuevo código es también MDS y para este código $b = d-1$ por lo que podríamos proceder como en el caso inicial. El protocolo que describiremos tendrá un cPoP de $b+k$.

Supondremos también una representación de los datos como en el modelo inicial. La dimensión del protocolo será $\rho = k$, al igual que en el anterior, y no necesitaremos suponer que los archivos están divididos en bandas, por lo que $\alpha = 1$. Por tanto podemos simplificar la notación de los fragmentos en los que está dividido un archivo ignorando el subíndice correspondiente a la banda: $x^i_{1j} \to x^i_j$. Este valor sería el fragmento $j$ del archivo $i$. Añadiremos también una tabla de valores

| **Variable** | **Significado** |
|---|---|
| $X^i$ | Archivo $i$ sin codificar, podemos representarlo como una matriz $\alpha \times k$ |
| $\mathcal{X}$ | Matriz de archivos formada por todos los $X^i$ |
| $\alpha$ | Número de bandas en las que está dividido cada archivo. |
| $C$ | Código MDS usado para codificar los archivos en los servidores |
| $k$ | Dimensión de $C$ |
| $n$ | Número de servidores y longitud de $C$ |
| $d$ | Distancia mínima de $C$, como suponemos $C$ MDS, $d = n-k+1$ |
| $b$ | Número de servidores cooperantes |
| $G$ | Matriz Generatriz de $C$, en forma sistemática |
| $H$ | Matriz de control de $C$ |
| $Y^i$ | Cada uno de los archivos codificados en los servidores, $Y^i = X^i G$ |
| $m$ | Número de archivos en la base de datos |
| $f$ | Índice del archivo deseado |
| $w_l$ | Vector columna de datos contenidos en un servidor $l$ |
| $\rho$ | Número de solicitudes necesarias para obtener $X^f$ |
| $Q_l$ | Matriz solicitud para un servidor $l$. Cada fila es la sub-solicitud $q_{l,i}$ |
| $U_i$ | Matriz aleatoria de tamaño $m \times (d-1)$ generada para la sub-solicitud i. |
| $Q'_i$ | $U_i H$, cada fila es una palabra de $C^\perp$. Cada columna es $q'_{j,i}$ |
| $r_{li}$ | Respuesta del servidor $l$ a la sub-solicitud $i$ |

Sea $f$ el índice del archivo que queremos obtener. Como tenemos que obtener $k$ valores para recuperar el archivo y el protocolo usa un total de $\rho = k$ sub-solicitudes, nos vale con explicar como obtener uno de los valores en cada sub-solicitud.

Sean $G$ la matriz generatriz del código (de tamaño $k \times n$) y $H$ la matriz de control (de tamaño $d-1 \times n$). Para cada sub-solicitud $i$ generamos $d-1$ vectores columna aleatorios $u_{1,i}, \ldots, u_{d-1,i}$ de longitud $m$. Agrupamos todos estos vectores en la matriz de dimensión $m \times d-1$,

$$U_i = \begin{pmatrix} u_{1,i} & u_{2,i} & \ldots & u_{d-1,i} \end{pmatrix}$$

Multiplicando ahora la matriz $U_i$ por $H$ generamos una matriz $Q'_i$ de tamaño $m \times n$. Como $H$ es la matriz generatriz del código dual, cada fila de $Q'_i$ es un elemento de este código dual, algo que posteriormente usaremos para conseguir recuperar valores de forma privada. La matriz $Q'_i$ es de la forma

$$U_i H = Q'_i = \begin{pmatrix} q'_{1,i} & q'_{2,i} & \ldots & q'_{n,i} \end{pmatrix}$$

donde cada $q'_{j,i}$ es un vector columna de tamaño $m$.

La i-ésima sub-solicitud que enviaremos a un servidor $l$ y por tanto la i-ésima fila de la matriz de solicitud $Q_l$ será

$$q_{l,i} = \begin{cases} q'_{l,i} + e_f & \text{si } l = i \\ q'_{l,i} & \text{en caso contrario} \end{cases}$$

donde $e_f$ es el vector canónico con un 1 en la posición $f$.

**Proposición 2.10.** *Podemos decodificar el archivo deseado $f$ a partir de las respuestas de los servidores a nuestras solicitudes $Q_l$*

*Demostración.* La respuesta de un servidor $l$ a una sub-solicitud $i$ será $r_{l,i} = q_{l,i}^T w_l$. Veremos que en la sub-solicitud $i$ podemos conseguir el fragmento $x_i^f$ del archivo $f$. De hecho obtendremos que este fragmento es la suma de todos los valores recibidos correspondientes a la i-ésima sub-solicitud:

$$x_i^f = \sum_{l=1}^{n} r_{l,i}$$

Para ver esto desarrollaremos la suma anterior:

$$\sum_{l=1}^{n} r_{l,i} = \sum_{l=1}^{n} q_{l,i}^T w_l = e_f w_i + \sum_{l=1}^{n} {q'}_{l,i}^T w_l$$

Denominamos ahora $(q'_{l,i})_j$ a la componente $j$ del vector $q'_{l,i}$. Entonces

$$e_f w_i + \sum_{l=1}^{n} {q'}_{l,i}^T w_l = e_f w_i + \sum_{l=1}^{n} \sum_{j=1}^{m} ({q'}_{l,i})_j (w_l)_j =$$

$$= e_f w_i + \sum_{j=1}^{m} \sum_{l=1}^{n} ({q'}_{l,i})_j (w_l)_j = e_f w_i + \sum_{j=1}^{m} 0 = x_i^f$$

La penúltima igualdad viene de que $\sum_{l=1}^{n} ({q'}_{l,i})_j (w_l)_j$ es el producto escalar de la fila $j$ de la matriz $Q'_i$ y la fila $j$ de $\mathcal{X}G$ que se corresponde a la codificación del archivo $j$ en los $n$ servidores. Como habíamos dicho anteriormente cada fila de $Q'_i$ es una palabra en el código dual y cada fila de $\mathcal{X}G$ es una palabra del código, por lo que el producto entre ambas es 0.

De esta forma con las $k$ sub-solicitudes obtenemos los $k$ fragmentos del archivo deseado almacenados en los $k$ primeros servidores. Como el código era sistemático estos fragmentos componen el archivo total que hemos sido capaces de recuperar.

Una forma más compacta de hacer esta demostración es la sucesión de igualdades siguiente:

$$\sum_{l=1}^{n} r_{l,i} = tr({Q'}_i^T \mathcal{X} G) + e_f w_i = tr(Q'_i G^T \mathcal{X}^T) + x_i^f = tr(U_i H G^T \mathcal{X}^T) + x_i^f = x_i^f$$

Recordamos que $tr$ representa a la traza de una matriz, es decir, la suma de todos los elementos de la diagonal. La primera igualdad viene de la definición de los $r_i$, la segunda de que la traza no varía transponiendo su interior, la tercera de la definición de $Q'_i$ y la última de que $HG^T = 0$ por la proposición 1.29.

□

**Ejemplo 2.11.** Consideramos una base de datos formada por 6 archivos $X^1 \dots X^6$ codificados en 5 servidores usando un código MDS de tipo $[2,5,4]$ sobre $\mathbb{F}_5$ con matriz generatriz

$$G = \begin{pmatrix} 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 2 & 3 \end{pmatrix}$$

Construiremos el protocolo descrito en esta sección para obtener el archivo $f = 2$ suponiendo que hay $b = 3$ servidores cooperantes. Recordamos que el protocolo tiene dimensión $k$ y por tanto tendremos que hacer $k$ sub-solicitudes. Construiremos únicamente la primera de ellas, ya que el resto se realiza de forma análoga. Inicialmente construimos 3 vectores columna aleatorios de dimension 6 que agrupamos en una matriz $U_1 = (u_{1,1}, u_{1,2}, u_{1,3})$. La matriz de control y por tanto generatriz del código dual será usando la proposición 1.30:

$$H = \begin{pmatrix} -1 & -1 & 1 & 0 & 0 \\ -1 & -2 & 0 & 1 & 0 \\ -1 & -3 & 0 & 0 & 1 \end{pmatrix}$$

Para obtener cada sub-solicitud tenemos que computar los $q'_{1,i}$ que son las columnas de $U_iH$. De hecho $q'_{1,i}$ será el producto de $U_i$ por la i-ésima columna de $H$. Así obtenemos que la primera sub-solicitud enviada a cada nodo será la siguiente:

$$\begin{aligned} q_{1,1} &= -u_{1,1} - u_{1,2} - u_{1,3} + e_2 \\ q_{1,2} &= -u_{1,1} - 2u_{1,2} - 3u_{1,3} \\ q_{1,3} &= u_{1,1} \\ q_{1,4} &= u_{1,2} \\ q_{1,5} &= u_{1,3} \end{aligned}$$

Una vez que hayamos recibido la respuesta $r_{1,i}$ de cada servidor $i$ podemos empezar con la decodificación. Tendremos el siguiente sistema de ecuaciones donde $I_{l,i} = u_{1,i}w_l$:

$$\begin{aligned} -I_{1,1} - I_{1,2} - I_{1,3} + x_1^2 &= r_{1,1} \\ -I_{2,1} - 2I_{2,2} - 3I_{2,3} &= r_{1,2} \\ I_{1,1} + I_{2,1} &= r_{1,3} \\ I_{1,2} + 2I_{2,2} &= r_{1,4} \\ I_{1,3} + 3I_{2,3} &= r_{1,5} \end{aligned}$$

Si sumamos todas las ecuaciones tendremos que $r_{1,1} + r_{1,2} + r_{1,3} + r_{1,4} + r_{1,5} = x_1^2$ de donde podemos sacar el fragmento del archivo.

**Ejemplo 2.12.** Consideraremos en este ejemplo el mismo caso que en el anterior pero únicamente para 2 servidores cooperantes. En este caso necesitaremos únicamente mandar solicitudes a $k+b=4$ de los 5 servidores. Por ello podemos reducir la longitud de nuestro código en 1, eliminando la última coordenada de cada palabra código. De esta forma el código resultante seguiría siendo un código MDS con matriz generatriz y de control:

$$G' = \begin{pmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 2 \end{pmatrix} \quad H' = \begin{pmatrix} -1 & -1 & 1 & 0 \\ -1 & -2 & 0 & 1 \end{pmatrix}$$

Esta vez solamente necesitaremos dos vectores aleatorios. De la misma forma que antes, la primera sub-solicitud a cada servidor serán:

$$\begin{aligned} q_{1,1} &= -u_{1,1} - u_{1,2} + e_2 \\ q_{1,2} &= -u_{1,1} - 2u_{1,2} \\ q_{1,3} &= u_{1,1} \\ q_{1,4} &= u_{1,2} \end{aligned}$$

Por lo que agrupando las respuestas en un sistema lineal obtendremos:

$$\begin{aligned} -I_{1,1} - I_{1,2} + x_1^2 &= r_{1,1} \\ -I_{2,1} - 2I_{2,2} &= r_{1,2} \\ I_{1,1} + I_{2,1} &= r_{1,3} \\ I_{1,2} + 2I_{2,2} &= r_{1,4} \end{aligned}$$

De aquí podemos obtener el fragmento del archivo sumando todas las respuestas igual que antes.

**Teorema 2.13.** *El protocolo descrito en esta sección garantiza PIR teórico de la información para $b \leq d-1$ servidores cooperantes con un cPoP de $b+k$*

*Demostración.* En primer lugar veremos que garantiza PIR teórico de la información. Tenemos que ver que $H(f|Q_j, j \in J) = H(f)$ para todos los conjuntos $J \subseteq [n]$, $|J| = b$. Para simplificar denominaremos $Q_J$ al conjunto de solicitudes $Q_j, j \in J$. Llamaremos también $U_I$ al conjunto de matrices aleatorias $U_i i = 1 \ldots k$ que hemos definido al principio del protocolo.

En primer lugar podemos tomar entropía condicionada tanto sobre $f$ como sobre $Q_J$ para obtener $H(Q_J) + H(f|Q_J) = H(f) + H(Q_J|f)$ Simplemente reorganizando los términos y operando obtenemos

$$\begin{aligned} H(f|Q_J) &= H(f) + H(Q_J|f) - H(Q_J) && (2.1) \\ &= H(f) - H(Q_J) + H(Q_J|f) - H(Q_J|f, U_I) && (2.2) \\ &= H(f) - H(Q_J) + I(Q_J, U_I|f) && (2.3) \\ &= H(f) - H(Q_J) + H(U_I|f) - H(U_I|Q_J, f) && (2.4) \\ &= H(f) - H(Q_J) + H(U_I) && (2.5) \\ &= H(f) && (2.6) \end{aligned}$$

Inicialmente usamos la definición de entropía conjunta y su simetría. En 2.2 el último término le podemos añadir porque como $Q_J$ está determinada por $f$ y las $U_I$, la variable aleatoria toma un único valor y su entropía es 0. En las dos igualdades siguientes aplicamos la definición de información mutua condicionada. En 2.4 usamos que las $U_I$ y $f$ son independientes por lo que $U_I|f \sim U_I$. También usamos que podemos usar el código MDS para obtener las $U_I$ a partir de $f$ y los $Q_J$ por lo que el sumando final es 0. Por último usamos que tanto $Q_J$ como $U_I$ siguen la misma distribución uniforme. Esto se debe a que ambas están formadas por $k$ matrices de $m(d-1)$ símbolos la primera y$d-1$ matrices de $km$ símbolos. Por tanto cada grupo de matrices está formado por la misma cantidad de símbolos y todas las posibles combinaciones tienen la misma probabilidad. Por ello lo que tienen la misma entropía. De esta forma llegamos al resultado que buscábamos.

En segundo lugar calcularemos el cPoP de este protocolo. Queremos descargar los $k$ fragmentos del archivo que deseamos. Para hacerlo nos descargamos un total de $b+k$ fragmentos por cada sub-solicitud. Como tenemos $k$ sub-solicitudes, eso hace un total de $(b+k)k$ fragmentos. El cPoP lo obtendremos dividiendo el segundo entre el primero haciendo un total de $b+k$. □

## 2.4. Otro protocolo PIR más eficiente

En esta última sección daremos un último protocolo para mejorar el cPoP en alguno de los casos de la sección anterior. Definiremos $\delta = \lfloor \frac{n-b}{k} \rfloor$. Consideraremos los casos en los que $\delta$ es distinto de 0. Vemos que para que esto ocurra, $n-b \geq k$ por lo que $b \leq d-1$ como en el caso anterior. La cota superior para el número de servidores cooperantes es la misma que en el caso anterior. De hecho para los casos $\delta = 1$ el protocolo será exactamente el mismo que el anterior. En el caso contrario lo que haremos será crear distintos grupos de servidores. En cada grupo de ellos aplicaremos el protocolo anterior y luego pondremos en común los resultados.

Solo necesitaremos enviar solicitudes a los $b+\delta k$ primeros servidores. Así que consideraremos a partir de ahora únicamente estos. Los últimos $b$ servidores serán servidores comunes y a los $\delta k$ restantes los dividimos en $\delta$ agrupaciones de $k$ servidores. Cada grupo de nodos en los que aplicaremos el protocolo anterior estará formado por los $b$ servidores comunes y una de las agrupaciones de $k$ servidores. Es decir el grupo $0 \leq g < \delta$ estará formado por los servidores con índices $I_g = \{gk+1, \ldots, (g+1)k\} \cup \{\delta k+1, \ldots, \delta k+b\}$. Lo haremos de la siguiente forma.

Para cada grupo $g$ creamos un nuevo código a partir del original. Este nuevo código será la reducción del original a los servidores del grupo, es decir, tendrá como matriz generatriz $G_g$ la sub-matriz formada por las columnas con índice en $I_g$. Podemos dividir esta matriz en dos partes, la primera, $B_g$, formada por las columnas de $G$ correspondientes la agrupación de $k$ servidores y la segunda, $P$, formada por las columnas de $G$ correspondientes a los $b$ servidores comunes. Así podemos representar cada una de las $G_i$ de la forma siguiente

$$G_i = \left( \begin{array}{c|c} B_g & P \end{array} \right)$$

El único problema es que es muy probable que estas matrices generatrices que hemos creado no sean sistemáticas. Sin embargo podemos solucionar el problema multiplicando

cada una de ellas por su correspondiente matriz $B_g^{-1}$. Sabemos que $B_g$ va a ser invertible ya que la matriz generatriz del código original $G$ es de rango $k$ por lo que todo subconjunto de $k$ columnas es linealmente independiente. Además podemos multiplicar ambas matrices ya que por la proposición 1.22 el producto de ambas genera el mismo código. Por ello tendremos para cada grupo una matriz generatriz en forma sistemática del código asociado a ese grupo. Así podemos representar tanto la matriz generatriz como la matriz de control de cada grupo, donde $I_k$ denota la identidad de dimension $k$:

$$G'_i = \left( \begin{array}{c|c} I_k & P'_g \end{array} \right) \quad H'_i = \left( \begin{array}{c|c} -P'^T_g & I_b \end{array} \right)$$

Por tanto hemos obtenido una matriz generatriz en forma sistemática del código asociado a cada uno de los grupos y su matriz de control. Cada uno de dichos códigos es un código MDS de tipo $[k+b, k, b+1]$ por lo que podemos aplicar el protocolo anterior para obtener un fragmento del archivo en cada sub-solicitud. Para ello nos basta con generar $b = d-1$ vectores aleatorios por sub-solicitud comunes que podemos usar para generar las sub-solicitudes correspondientes de cada uno de los grupos de servidores.

**Teorema 2.14.** *Usando el protocolo descrito en esta sección podemos recuperar el archivo deseado*

*Demostración.* Como explicamos al principio de la sección, hemos creado $\delta$ agrupaciones de $b+k$ servidores y hemos construido una matriz generatriz sistemática de cada uno de los códigos. Usando el protocolo de la sección anterior en cada grupo de servidores, podemos recuperar de forma privada un fragmento por agrupación por sub-solicitud. Lo que hace un total de $\delta$ fragmentos por sub-solicitud. Así necesitaremos un total de $k/\delta$ sub-solicitudes para recuperar $k$ fragmentos del archivo deseado con los que lo podemos reconstruir. □

**Ejemplo 2.15.** En el siguiente ejemplo tenemos una base de datos formada por 9 servidores con los datos codificados usando un código MDS de parámetros $[2, 9, 8]$ sobre el cuerpo $\mathbb{F}_{11}$. Queremos realizar el protocolo PIR descrito en esta sección para el caso de $b = 3$ cooperantes, recuperando el archivo número 2 en una base de datos con tres archivos. Sea la matriz generatriz del código la siguiente:

$$G = \begin{pmatrix} 1 & 0 & 10 & 9 & 8 & 7 & 6 & 5 & 4 \\ 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \end{pmatrix}$$

En este caso $\delta = \lfloor (9-3)/2 \rfloor = 3$ por lo que tendremos tres agrupaciones de servidores. Si tomamos los últimos $b = 3$ servidores como comunes, las agrupaciones de servidores serán las siguientes (representamos cada servidor por su indice $1 \leq i \leq 9$):

$$I_0 = \{1, 2, 7, 8, 9\}$$
$$I_1 = \{3, 4, 7, 8, 9\}$$
$$I_2 = \{5, 6, 7, 8, 9\}$$

En cada una de las agrupaciones aplicaremos el protocolo de la sección anterior usando las matrices restringidas a dichos servidores. Realizaremos el caso de la segunda agrupación

$I_1$. La matriz generatriz restringida a dichos servidores será correspondiente a las columnas con índices en $I_1$. Esta es

$$G_1 = \begin{pmatrix} 10 & 9 & 6 & 5 & 4 \\ 2 & 3 & 6 & 7 & 8 \end{pmatrix}$$

Claramente esta matriz no está en forma sistemática por lo que no podemos aplicar el protocolo aún. Para transformarla en una sistemática debemos de multiplicarla por la inversa de la submatriz formada por las dos primeras columnas. La matriz resultante $G'_1$ generará el mismo código y será sistemática:

$$G'_1 = \begin{pmatrix} 3 & 2 \\ 9 & 10 \end{pmatrix} \begin{pmatrix} 10 & 9 & 6 & 5 & 4 \\ 2 & 3 & 6 & 7 & 8 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 8 & 7 & 6 \\ 0 & 1 & 4 & 5 & 6 \end{pmatrix}$$

Realizando el mismo procedimiento con las otras agrupaciones obtenemos las siguientes matrices generatrices:

$$G'_0 = \begin{pmatrix} 1 & 0 & 6 & 5 & 4 \\ 0 & 1 & 6 & 7 & 8 \end{pmatrix} \quad G'_2 = \begin{pmatrix} 1 & 0 & 10 & 9 & 8 \\ 0 & 1 & 2 & 3 & 4 \end{pmatrix}$$

Ahora con la matriz en la forma deseada podemos aplicar el protocolo anterior y obtener de forma privada partes del archivo. Para ello necesitamos las matrices de control:

$$H'_0 = \begin{pmatrix} 5 & 5 & 1 & 0 & 0 \\ 4 & 6 & 0 & 1 & 0 \\ 3 & 7 & 0 & 0 & 1 \end{pmatrix} \quad H'_1 = \begin{pmatrix} 3 & 7 & 1 & 0 & 0 \\ 4 & 6 & 0 & 1 & 0 \\ 5 & 5 & 0 & 0 & 1 \end{pmatrix} \quad H'_2 = \begin{pmatrix} 1 & 9 & 1 & 0 & 0 \\ 2 & 8 & 0 & 1 & 0 \\ 3 & 7 & 0 & 0 & 1 \end{pmatrix}$$

Ahora para aplicar el protocolo del apartado anterior generamos una matriz aleatoria de $m \times (d-1) = 3 \times 7$ y multiplicándola por cada una de las matrices de control, obteniendo:

$$U = \begin{pmatrix} 2 & 2 & 3 \\ 4 & 4 & 6 \\ 7 & 8 & 9 \\ 10 & 0 & 3 \\ 4 & 5 & 1 \\ 1 & 2 & 4 \\ 2 & 2 & 2 \end{pmatrix}$$

$$Q_0 = \begin{pmatrix} 5 & 10 & 2 & 2 & 3 \\ 10 & 9 & 4 & 4 & 6 \\ 6 & 3 & 7 & 8 & 9 \\ 4 & 5 & 10 & 0 & 3 \\ 10 & 2 & 4 & 5 & 1 \\ 3 & 1 & 1 & 2 & 4 \\ 2 & 3 & 2 & 2 & 2 \end{pmatrix} \quad Q_1 = \begin{pmatrix} 7 & 8 & 2 & 2 & 3 \\ 3 & 5 & 4 & 4 & 6 \\ 10 & 10 & 7 & 8 & 9 \\ 1 & 8 & 10 & 0 & 3 \\ 4 & 8 & 4 & 5 & 1 \\ 9 & 6 & 1 & 2 & 4 \\ 2 & 3 & 2 & 2 & 2 \end{pmatrix} \quad Q_2 = \begin{pmatrix} 4 & 0 & 2 & 2 & 3 \\ 8 & 0 & 4 & 4 & 6 \\ 6 & 3 & 7 & 8 & 9 \\ 8 & 1 & 10 & 0 & 3 \\ 6 & 6 & 4 & 5 & 1 \\ 6 & 9 & 1 & 2 & 4 \\ 1 & 4 & 2 & 2 & 2 \end{pmatrix}$$

Como es lógico, las tres últimas columnas, que son las correspondientes a los servidores comunes son iguales y por ello podremos usar su respuesta en la decodificación de los fragmentos en cualquiera de los tres grupos.

**Teorema 2.16.** *El protocolo garantiza PIR teórico de la información para b servidores cooperantes con un cPoP de* $\frac{b+\delta k}{\delta}$.

*Demostración.* La sub-solicitud $i$ que enviamos a cada grupo de servidores están generadas a partir de las columnas del producto de la matriz aleatoria $U_i$ y cada una de las matrices de control. Sin embargo todas estas las podemos agrupar en una única matriz $H$ y así obtener todas a la vez:

$$H = \begin{pmatrix} -P'^T_1 & -P'^T_2 & \dots & -P'^T_\delta & I_b \end{pmatrix}$$

El dual de este código tiene como matriz generatriz:

$$G = \begin{pmatrix} & P_1 \\ & P_2 \\ I_{\delta k} & \vdots \\ & P_\delta \end{pmatrix}$$

Esta matriz genera un código MDS de tipo $[\delta k + b, \delta k, b + 1]$ por lo que el código generado por $H$ es de tipo $[\delta k + b, b, \delta k + 1]$. Esto quiere decir que cualesquiera $b$ sub-solicitudes son linealmente independientes por lo que se garantiza la privacidad para $b$ servidores cooperantes.

Por último en cada sub-solicitud el usuario descifra un fragmento por cada grupo haciendo un total de $\delta$ fragmentos. Para ello recibe una respuesta de $\delta k + b$ servidores. Independientemente del número de sub-solicitudes el cPoP será de:

$$cPoP = \frac{b + \delta k}{\delta}$$

□

# Capítulo 3

# El producto estrella y su utilidad en los protocolos PIR

En el capítulo anterior hemos visto protocolos que nos permiten recuperar información de una base de datos distribuida de forma bastante eficiente, especialmente en el caso de $b = 1$ con el protocolo de la sección 2.2, y el caso $b = d - 1$ con el protocolo de la sección 2.3. Sin embargo para los casos con un número de servidores cooperantes entre estos dos valores, muchas veces el protocolo de la sección 2.4 no supone una mejora al de la sección anterior y en estos casos, este último no tiene un cPoP demasiado bueno, ya que el propio protocolo no realiza solicitudes para los últimos $n - b$ servidores. Por ello nos gustaría expandir los protocolos de la sección 2.2 y 2.3 para un número intermedio de servidores cooperantes de forma que las solicitudes se hagan para todos los servidores y se obtenga un mejor cPoP.

Otro problema que nos surge con los protocolos explicados en el tema anterior es que únicamente son eficaces cuando el código con el que se realiza la encriptación en los servidores es un código MDS, de forma que cuando el código usado no es de este tipo, no tenemos forma de recuperar la información.

En este capítulo introduciremos una nueva operación sobre códigos llamada el producto estrella ($\star$). Veremos qué propiedades tiene esta operación y como podemos usar estas mismas para definir un nuevo protocolo que nos permita solucionar los problemas explicados anteriormente. Además veremos como podemos aplicar también el producto estrella para ser capaz de recuperar un archivo en la presencia de servidores bizantinos y no responsivos. Este nuevo protocolo apareció publicado en [7].

## 3.1. Producto estrella

El producto estrella o producto de Schur es una operación entre subespacios vectoriales definida por el matemático Issai Schur en [8]. Dado que un código es también un subespacio vectorial el resultado aplica también a estos. El producto estrella ha encontrado varias aplicaciones en la teoría de códigos ya que, como veremos, tiene propiedades muy útiles.

**Definición 3.1.** Sea $\mathbb{F}_q^n$ un espacio vectorial. Entonces el producto de Hadamard entre dos vectores $v = [v_1, \ldots, v_n]$ $w = [w_1, \ldots, w_n]$ se define como el producto componente a

componente de los dos vectores, es decir, $v \star w = [v_1w_1, \ldots, v_nw_n]$.

**Definición 3.2.** Sean $V, W$ dos subespacios vectoriales de $\mathbb{F}_q^n$. Entonces definimos el producto estrella (o de Shur) de $V$ y $W$ como el subespacio generado por el producto de Hadamard entre los elementos de $V$ y $W$.

$$V \star W = span(\{v \star w | v \in V, w \in W\})$$

**Proposición 3.3.** *El producto estrella de dos códigos $C$, $D$ sobre $\mathbb{F}_q^n$ cumple las siguientes propiedades:*

1. *Si $D$ es el código de repetición de longitud $n$, $Rep(n)_q$, es decir, el código formado por las palabras con todas las componentes iguales, entonces $C \star D = C$*
2. *Si $(C \star D)^\perp = E$ para cierto código $E$, entonces $d_E \geq d_{C^\perp} + d_{D^\perp} - 2$ donde $d_C$ es la distancia mínima del código $C$.*
3. *Si $C$ es un código MDS, entonces $(C \star C^\perp)^\perp = Rep(n)_q$*
4. *Si $C$ y $D$ son códigos GRS con parámetros $C = GRS_{k_1}(\alpha, v_1)$, $D = GRS_{k_2}(\alpha, v_2)$. Entonces $C \star D = GRS_{min\{k_1+k_2-1,n\}}(\alpha, v_1 \star v_2)$ .*

*Demostración.*

1. Sean $c \in C, d = (x, \ldots, x) \in D$, entonces $c \star d = x \cdot c \in C$ por lo que $C \star D \subseteq C$. De forma similar $c = (x^{-1} \cdot c) \star d$ ya que $(x^{-1} \cdot c) \in C$ y obtenemos la otra contención y por tanto la igualdad.
2. La prueba de esta proposición es extensa y se necesitan nuevos conceptos y por brevedad se omitirá. Para más información es una aplicación directa del teorema 5 de [9].
3. Sea $(C \star C^\perp)^\perp = E$. Tomando el duales obtenemos $C \star C^\perp = E^\perp$. Aplicando la desigualdad del apartado anterior tenemos que $d_{E^\perp} \geq d_{C^\perp} + d_C - 2 = n - (n-k) + 1 + n - k + 1 - 2 = n$. Como $d_{E^\perp} \leq n$ tenemos que $d_{E^\perp} = n$. Usando la desigualdad de Singleton entonces $k = n - d + 1 = 1$. El único código de distancia $n$ y dimensión 1 es $Rep(n)_q$.
4. Sean $c \in C, d \in D$. Sea $E = GRS_{min\{k_1+k_2-1,n\}}(\alpha, v_1 \star v_2)$. Por ser códigos GRS, $c = (v_{11}f_c(\alpha_1), \ldots, v_{1n}f_c(\alpha_n))$ y también $d = (v_{21}f_d(\alpha_1) \ldots v_{2n}f_d(\alpha_n))$. Por tanto $c \star d = (v_{11}v_{21}f_cf_d(\alpha_1), \ldots, v_{1n}v_{2n}f_cf_d(\alpha_n))$. Como $deg(f_c(x)f_d(x)) \leq k_1 - 1 + k_2 - 1 < k_1 + k_2 - 1$ tenemos que $c \star d \in E$. El mínimo entre $n$ y el máximo grado posible se debe a que en caso de que el grado del polinomio producto sea mayor, siempre podremos encontrar otro polinomio $h$ de grado $n$ tal que en todos los puntos de $\alpha$ se cumpla $h(\alpha_i) = f_cf_d(\alpha_i)$. Con esto obtenemos la contención $C \star D \subseteq E$. Para la otra usaremos que $E$ está generado por las palabras $(v_{11}v_{21}h(\alpha_1), \ldots, v_{1n}v_{2n}h(\alpha_n))$ donde $h$ es una potencia de $x$, $h(x) = x^m$, con $m \leq k_1 + k_2 - 1$. Podemos obtener todas estas palabras tomando palabras en $C$ y $D$ generados por polinomios $f_c = x^a, f_d = x^b$ de forma que $m = a + b$.

□

## 3.2. Protocolo

Como hemos visto, el producto estrella tiene unas propiedades muy interesantes, especialmente en códigos MDS y GRS que podemos usar en nuestro favor. En esta sección describiremos un protocolo que usa estas propiedades para conseguir PIR en una base de datos codificada por un código $C$ arbitrario con matriz generatriz $G$. La idea principal de este nuevo protocolo es la de usar un código $D$ como fuente de dónde sacar vectores aleatorios que actúan como "ruido". Durante la fase de recuperación de los datos, las respuestas de los servidores serán la suma de palabras de $C \star D$ y algún valor de los que queremos recuperar. Tanto el cPoP como la cantidad de servidores cooperantes contra los que nuestro protocolo protege, dependerá de la elección de este código $D$. Al igual que en el capítulo anterior daremos una tabla de parámetros para facilitar la comprensión del protocolo:

| **Variable** | **Significado** |
|---|---|
| $X^i$ | Archivo $i$ sin codificar, podemos representarlo como una matriz $\alpha \times k$ |
| $\alpha$ | Número de bandas en las que está dividido cada archivo. |
| $C$ | Código MDS usado para codificar los archivos en los servidores |
| $k$ | Dimensión de $C$ |
| $n$ | Número de servidores y longitud de $C$ |
| $d$ | Distancia mínima de $C$ |
| $b$ | Número de servidores cooperantes |
| $G$ | Matriz Generatriz de $C$ |
| $D$ | Código que usaremos para crear la parte aleatoria de las solicitudes |
| $c$ | Distancia mínima de $C \star D$. Número de símbolos recuperados por sub-solicitud |
| $Y^i$ | Cada uno de los archivos codificados en los servidores, $Y^i = X^i G$ |
| $m$ | Número de archivos en la base de datos |
| $f$ | Índice del archivo deseado |
| $w_l$ | Vector columna de datos contenidos en un servidor $l$ |
| $\rho$ | Número de solicitudes necesarias para obtener $X^f$ |
| $d_j$ | Vector fila formado por las componentes j-ésimas de cada palabra aleatoria $d^{p,a}$ |
| $J_i$ | Servidores de donde recuperaremos información en la sub-solicitud $J$ |
| $J_i^a$ | Elemento $a$ de la partición de $J_i$ |
| $g$ | Cantidad de símbolos de una banda descargados en una sub-solicitud, $g = c/\alpha$ |
| $Q_l$ | Matriz solicitud para un servidor $l$. Cada fila es la sub-solicitud $q_{l,i}$ |
| $r_{li}$ | Respuesta del servidor $l$ a la sub-solicitud $i$ |
| $r_i$ | Agrupación de las $n$ respuestas de los servidores a la primera sub-solicitud |

Supongamos por tanto una base de datos distribuida en la que hay un total de $m$ archivos codificados usando un código $C = [n, k, d]$ sobre $\mathbb{F}_q$ de forma similar a cómo explicamos en capítulo anterior. En este caso no hace falta que asumamos que la matriz generatriz tiene una forma sistemática. Por ello, denotaremos $X^i = [x^i_{a,j}]_{a \in \alpha, j \in k}$ a cada archivo $i \in [m]$ que queremos codificar en los $n$ servidores y llamaremos $Y^i = X^i G = [y^i_{a,j}]_{a \in \alpha, j \in n}$ a los archivos ya codificados en la base de datos distribuida usando el código $C$.

Para obtener la parte aleatoria de las sub-solicitudes usaremos un código $D$. En principio la única condición que debe cumplir $D$ es ser de longitud $n$, pero la cantidad de servidores

contra los que el protocolo protege y el cPoP dependerá de como elijamos este código. Definimos $c = d_{C\star D} - 1$. En cada sub-solicitud seremos capaces de recuperar $c$ símbolos del archivo que deseamos $f$. Necesitamos entonces para la mayor eficiencia posible que los símbolos que descargamos en $\rho$ solicitudes sean exactamente los $\alpha k$ símbolos a partir de los cuales podemos recuperar el archivo. Por ello, $\alpha k = \rho c$. De esta forma necesitamos que los archivos de la base de datos estén divididos en $\alpha = \dfrac{mcm(c,k)}{k}$ bandas y nuestro protocolo usará un total de $\rho = \dfrac{mcm(c,k)}{c}$ sub-solicitudes.

Como hemos dicho, en cada sub-solicitud recuperaremos $c$ símbolos del archivo deseado. Todos los símbolos que recuperemos del archivo proveerán de un mismo subconjunto $J \subseteq [n]$ de los servidores. El tamaño de $J$ será

$$|J| = max\{c, k\}$$

Podemos usar cualquier subconjunto de este tamaño de los servidores pero por simplicidad supondremos que son los primeros. En caso contrario simplemente podríamos reordenarlos para que sea el caso. Por tanto $J = [1, \dots, |J|]$. Que solo obtengamos símbolos de estos servidores no quiere decir que en cada sub-solicitud recuperemos un símbolo de cada uno de ellos, de hecho, en cada sub-solicitud recuperaremos exactamente $c$ símbolos, aunque $|J| > c$. Para cada sub-solicitud $i$ denotaremos como $J_i \subseteq J$ el conjunto de servidores de los que recibiremos un símbolo de $f$ en dicha sub-solicitud.

A continuación indicaremos cuales serán las sub-solicitudes que se enviarán a cada servidor. Estas se definirán de forma recursiva a partir de la primera de ellas. Por ello daremos cual será la sub-solicitud inicial y como podremos obtener el resto a partir de ella.

Para empezar tomamos $m\alpha$ palabras de $D$ aleatorias $d^{p,a} = (d_1^{p,a}, \dots, d_n^{p,a})$, con $p \in [m], a \in [\alpha]$, es decir, una por cada banda que forma la base de datos. A partir de estas palabras código aleatorias, definiremos para cada $j \in [n]$ los vectores fila

$$d_j = (d_j^{1,1}, \dots, d_j^{1,\alpha}, \dots, d_j^{m,1}, \dots, d_j^{m,\alpha}) \in \mathbb{F}_q^{m\alpha}$$

es decir, el vector formado por las componentes j-ésimas de cada uno de las palabras código aleatorias.

En la primera sub-solicitud obtendremos símbolos del archivo $f$ a través de los primeros $c$ servidores, por tanto, $J_1 = [c]$. Ahora dividiremos $J_1$ en $\alpha$ partes de $g = c/\alpha = k/\rho$ elementos cada uno. De hecho de la definición de $\alpha$ podemos ver que $g$ es un número entero. Las partes serán las siguientes:

$$J_1^1 = \{1, \dots, g\}, J_1^2 = \{g+1, \dots, 2g\}, \dots\dots, J_1^\alpha = \{(\alpha-1)g+1, \dots, \alpha g = c\}$$

A partir de los vectores $d_j$ y estos grupos definimos la primera sub-solicitud que enviamos a cada servidor $l$. Separamos dos casos, el caso en el que $l$ pertenece a $J_1$ y por tanto también a alguno de los $J_1^a$ y el caso en el que no.:

$$q_{l,1} = \begin{cases} d_l + e_{\alpha(f-1)+a} & \text{si } l \in J_1^a \\ d_l & \text{si } l \notin J_1 \end{cases}$$

Para calcular las siguientes solicitudes, como hemos dicho, lo haremos recursivamente a partir de la anterior. En primer lugar generamos nuevos vectores aleatorios $d^{p,a}$ para

$p \in [m], a \in [\alpha]$ y obtenemos a partir de ellos los $d_j$. En segundo lugar, para cierta sub-solicitud $i$, definimos los conjuntos de donde obtendremos la información $J_i^a$ a partir de cada $J_{i-1}^a = \{j_1, \dots, j_g\}$ de la siguiente forma:

$$J_i^a = \{j_1 + g, \dots, j_g + g\}$$

donde si para algún índice $j_k + g \notin J$ entonces lo sustituiremos por $(j_k + g) \bmod |J|$. Podemos decir pues, que $J_i^a$ se calcula rotando cíclicamente sobre $J$ los elementos de $J_{i-1}^a$ un total de $g$ posiciones. Con estos conjuntos calculados podemos definir $J_i = J_i^1 \cup \cdots \cup J_i^\alpha$ y la sub-solicitud que enviaremos será de forma similar a el primer caso:

$$q_{l,i} = \begin{cases} d_l + e_{\alpha(f-1)+a} & \text{si } l \in J_i^a \\ d_l & \text{si } l \notin J_i \end{cases}$$

Ahora tenemos que ver cómo enviando estas solicitudes podremos recuperar el archivo que deseamos. Para ello tenemos que realizar un análisis de las respuestas que recibiremos de los servidores. Como en los capítulos anteriores, la respuesta a cada sub-solicitud será el producto de ésta por el vector de datos que cada servidor tiene almacenado, $r_{i,j} = q_{j,i} w_j$. Como hemos dividido las sub-solicitudes en dos grupos recibiremos dos tipos distintos de respuestas.

**Proposición 3.4.** *La agrupación de las respuestas de los $n$ servidores a la misma sub-solicitud $i$, $r_i = (r_{i,1}, r_{i,2}, \dots, r_{i,n})$, esta formada por la suma de una palabra de $C \star D$ más un vector $v_{i,J_i}$ con soporte únicamente en $J_i$.*

*Demostración.* En primer lugar, para un servidor $l \notin J_i$ la respuesta a la sub-solicitud i-ésima tendrá la siguiente forma:

$$r_{i,j} = q_{j,i} w_j = d_j w_j$$

Sin embargo, para los servidores $j \in J_i^a$ para cierto $a \in \alpha$, tendremos que

$$r_{i,j} = q_{j,i} w_j = d_j w_j + e_{\alpha(f-1)+a} w_j = d_j w_j + y_{a,j}^f$$

donde $y_{a,j}^f$ es el símbolo almacenado por el servidor $j$ del archivo $f$ en la banda $a$.

Si agrupamos las $n$ respuestas de los servidores tenemos que:

$$r_i = (d_j w_j)_{j \in [n]} + \sum_{j \in J_i} e_j y_{a,j}^f$$

El segundo sumando es claramente un vector con soporte únicamente en $J_i$, por lo que tenemos que ver que el primer sumando es una palabra de $C \star D$. Para ello usaremos la definición de $d_j$ en 3.2.

$$(d_j w_j)_{j \in [n]} = (\sum_{p=1}^{m} \sum_{a=1}^{\alpha} d_j^{p,a} y_{a,j}^p)_{j \in [n]} = \sum_{p=1}^{m} \sum_{a=1}^{\alpha} (d_j^{p,a} y_{a,j}^p)_{j \in [n]} = \sum_{p=1}^{m} \sum_{a=1}^{\alpha} d^{p,a} \star y_a^p$$

donde $y_a^p = (y_{a,1}^p, \dots y_{a,n}^p)$ es la banda $a$ del archivo $p$ de la base de datos distribuida.

Dado que cada $d^{p,a}$ es una palabra de $D$ y cada banda es una palabra de $C$ y la suma de dos palabras de un código sigue estando en el código, tenemos que efectivamente este primer sumando es una palabra de $C \star D$. $\square$

**Teorema 3.5.** *Es posible recuperar el archivo $f$ a partir de las respuestas de los $n$ servidores a las solicitudes $q_{j,i}$ para $j \in [n], i \in [\rho]$ si cualesquiera $k$ columnas de las $|J|$ iniciales de la matriz de control de $C$, $H$, son linealmente independientes.*

*Demostración.* Para cierta sub-solicitud i, hemos visto en la proposición anterior que $r_i = z_i + v_{i,J_i}$ donde el primer sumando es una palabra de $C \star D$ y el segundo tiene soporte únicamente en $J_i$. Sea $H$ una matriz de control de $C \star D$ o lo que es lo mismo, una matriz generatriz de su dual. Entonces

$$Hr_i = Hz_i + Hv_{i,J_i} = Hv_{i,J_i}$$

dado que una matriz de control de un código por un vector cualquiera del mismo es 0.

Dado que $H$ es una matriz de control de $C \star D$ y $c = d_{C\star D} - 1$ podemos aplicar la proposición 1.31 para decir que cualesquiera $c$ columnas de $H$ son linealmente independientes. Esto nos permite resolver el sistema lineal anterior y recuperar los valores $y^f_{a,j}$ que forman el vector $v_{i,J_i}$, con $j \in J_i^a$ para cada $a \in [\alpha]$.

En cada sub-solicitud recuperamos un total de $c$ símbolos del archivo codificado, así que en total recuperaremos $c\rho = \alpha k$ elementos. Si para cada banda $a \in [\alpha]$ recuperamos exactamente $k$ de ellos, entonces seremos capaz de usar el código $C$ para decodificar las palabras código y obtener el archivo original. Veremos ahora que efectivamente es el caso.

Sea $a \in \alpha$. Los símbolos de la banda $a$ que recuperaremos vendrán de los servidores con índices en algún $J_i^a$ para $i \in [\rho]$. Para cada uno de ellos $|J_i^a| = g$ y por tanto en el caso de ser todos disjuntos recibiremos $gs = k$ de ellos en total, tal y como queremos. De hecho

$$\bigcup_{i=1}^{\rho} J_i^a = \bigcup_{i=1}^{\rho} \{j_1 + (i-1)g, \ldots, j_1 + g - 1 + (i-1)g\} = \{j_1, \ldots, j_1 + g\rho - 1\} = \{j_1, \ldots, j_1 + k - 1\}$$

con la restricción de que si algún elemento es mayor que $J$ entonces tomaremos su clase de equivalencia modulo $|J|$. Dado que $|J| \geq k$ todos estos elementos serán disjuntos y podremos decodificar todas las bandas del archivo. $\square$

**Teorema 3.6.** *El protocolo PIR descrito en esta sección protege contra $b = d_{D^\perp} - 1$ servidores cooperantes.*

*Demostración.* Sea $T \subseteq [n]$ un subconjunto de $n$ servidores de tamaño $t \leq d_{D^\perp} - 1$. Primero probaremos la afirmación para una única sub-solicitud y después veremos que el resultado es el mismo para las $\rho$ a la vez.

En primer lugar, sabemos que $t \leq d_{D^\perp}$, por lo que aplicando la proposición 1.31 sabemos que cualesquiera $t$ columnas de la matriz generatriz de $D$ (que es la matriz de control de $D^\perp$) son linealmente independientes. Esto quiere decir que si restringimos el código $D$ a las componentes de $T$, nos queda un código $D_T$ que es el espacio total $\mathbb{F}_q^t$.

Cada sub-solicitud $q_{j,i}$ que hemos enviado a un servidor $j$ en la sub-solicitud $i$ es de la forma $d_j + e_l$ para cierto índice $l$. El vector $d_j$ lo habíamos obtenido a partir de los $\alpha m$ vectores aleatorios $d^{p,a}$ de la forma:

$$d_j = (d_j^{1,1}, \dots, d_j^{1,\alpha}, \dots, d_j^{m,1}, \dots, d_j^{m,\alpha}) \in \mathbb{F}_q^{1\times m\alpha}$$

Como cada vector $d^{p,a}$ se ha obtenido uniformemente de forma aleatoria sobre $D$, la restricción de estos vectores en $D_T$ también sigue una distribución uniforme, y como este último es el espacio total $\mathbb{F}_q^t$, cada una de estas componentes del vector está distribuida uniformemente sobre $\mathbb{F}_q$. Por tanto el vector $d_j$ también sigue una distribución uniforme sobre $\mathbb{F}_q^{1\times\alpha m}$. De la misma forma, como $D_T = \mathbb{F}_q^t$, la distribución $d_T = \{d_j | j \in T\}$ es uniforme sobre $(\mathbb{F}_q^{1\times\alpha m})^t$.

Por último $q_T = \{q_j | j \in T\} = \{d_j + e_l | j \in T\}$ seguirá la misma distribución que $d_T$ ya que es la misma más una constante. Esto quiere decir que para una $i$ fijo, las sub-solicitudes que enviamos $q_T$ a los servidores en $T$, siguen también una distribución uniforme. Esto quiere decir que su distribución es independiente del archivo que queremos recuperar $f$ y por tanto para una sub-solicitud fija , $H(q_T) = H(q_T|f)$.

Para el caso en el que consideramos todas las sub-solicitudes $Q_T$ y no solo una sub-solicitud, el razonamiento es el mismo. En cada sub-solicitud, los vectores aleatorios $d^{p,a}$ se escogen de forma independiente al resto de sub-solicitudes por lo que un razonamiento como antes nos lleva a ver que $Q_T$ sigue una distribución uniforme sobre $(\mathbb{F}_q^{1\times\alpha m})^{t\rho}$, llegando a la misma conclusión de $H(Q_T) = H(Q_T|f)$ □

**Proposición 3.7.** *El cPoP del protocolo explicado en esta sección es* $n/d_{C\star D} - 1$

*Demostración.* El total de símbolos que queremos obtener es $\alpha k$. En cada sub-solicitud descargamos $n$ símbolos, uno por cada servidor. Como el protocolo usa en total $\rho = k\alpha/c$ sub-solicitudes, en total descargamos $nk\alpha/c$ símbolos. Por tanto el cPoP es:

$$\frac{nk\alpha/c}{\alpha k} = \frac{n}{c} = \frac{n}{d_{C\star D} - 1}$$

□

Un problema que podemos encontrarnos en la aplicación de este tipo de protocolos en una situación real es que el número de bandas $\alpha$ en las que está dividido un archivo de la base de datos no es algo que nosotros podamos controlar, y por tanto en muchos casos, este número no será el requerido por el protocolo $\alpha'$. Sin embargo, este problema tiene una solución fácil. En el caso de que $\alpha < \alpha'$ la solución es usar el protocolo para obtener no solo las $\alpha$ bandas del archivo que deseamos, si no que también las $\alpha' - \alpha$ bandas del archivo siguiente. En el caso $\alpha > \alpha'$ podemos aplicar el protocolo para obtener las primeras $\alpha'$ bandas del archivo deseado y posteriormente aplicar otra vez más el protocolo para obtener las siguientes $\alpha'$ bandas y así sucesivamente hasta conseguir todas las bandas requeridas. Sin embargo, en ambos casos, este ajuste supone un incremento del cPoP necesario para descargar el archivo.

## 3.3. Particularidades en los códigos GRS

Como hemos visto, este protocolo puede ser aplicado para cualquier código $C$, independientemente de su distancia. Sin embargo, al final de la sección anterior vimos que tanto la cantidad de servidores contra los que se protege, como el cPoP de éste, depende de la distancia mínima en $C \star D$ y por tanto depende de las características del código inicial $C$ y de qué código elijamos para hacer las solicitudes $D$. Por eso sigue teniendo sentido en este caso tomar un código $C$ con una distancia mínima lo más grande posible, es decir, un código Reed-Solomon generalizado. En esta sección veremos que código $D$ podríamos tomar para que el protocolo proteja contra el mayor número de servidores posible y el cPoP sea bajo.

Los códigos GRS tienen dos propiedades que nos pueden resultar útiles a la hora de encontrar un código $D$ lo mejor posible. La primera, que ya demostramos en 3.3, es que el conjunto de códigos GRS asociados a una misma n-upla $\alpha$ es cerrado por el producto estrella. Además lo mismo ocurre tomando duales. La segunda propiedad es que aunque la dimensión de $C \star D$ para dos códigos arbitrarios puede ser hasta $dim(C) \cdot dim(D)$, si ambos son códigos GRS, esta dimensión únicamente puede llegar a $dim(C) + dim(D) - 1$. Esta última propiedad es una mejora significante sobre el resto de códigos que hará que la distancia mínima del producto estrella sea mayor. Sin embargo para probar estas afirmaciones necesitamos introducir dos lemas.

**Lema 3.8.** *Sean $C, D$ dos códigos lineales con soporte en $\{1 \dots n\}$. Sea $k_C$ la dimensión de $C$ y sea $d_D^\perp$ la distancia mínima del dual de $D$. Entonces:*

$$dim(C \star D) \geq min\{n, k_C + d_D^\perp - 1\}$$

*Demostración.* Sea $m = min\{n, k_C + d_D^\perp - 1\}$. Tenemos entonces que $d_D^\perp \geq m - k_1 + 1$, por lo que aplicando 1.31 tenemos que cualesquiera $m - k_1 + 1$ columnas de la matriz generatriz de $D$ son linealmente independientes. Esto quiere decir que para cualquier conjunto de índices $J$ de tamaño $m - k_1$ y otro índice $i \notin J$ podemos encontrar un vector $y$ de forma que $y$ sea 0 en las coordenadas en $J$ y en la coordenada $i$ sea 1.

Usaremos esto para obtener $m$ palabras de $C \star D$ linealmente independientes. En primer lugar sea $G_C$ la matriz generatriz de $C$ en forma sistemática. Para cada $i \in [m]$ necesitamos una palabra $z \in C \star D$ de forma que que $z$ sea 0 en las coordenadas en $[m]$
$\{i\}$. Para obtenerla diferenciamos dos casos.

Si $i \in [k_C]$, tomamos $x \in C$ la i-ésima fila de $G_C$ y tomamos $y \in D$ como la palabra de $D$ con 0 en las coordenadas $[m] \setminus [k_1]$. Entonces $x \star y$ cumple las condiciones requeridas.

En el caso contrario tomamos $x$ como una fila de $G_C$ que no se anula en la coordenada $i$ (tiene que haber al menos una porque $C$ tiene soporte $1 \dots n$). Sea esta fila la fila $j$. Tomaremos $y$ como el vector de $D$ que se anula en $[m] \setminus [k_C] \setminus \{i\} \cup \{j\}$. De la misma forma $x \star y$ tiene las propiedades que necesitamos. □

**Lema 3.9.** *Para dos códigos lineales cualesquiera $C, D$ con soporte en $n$ tenemos que:*

$$C \perp D \star (C \star D)^\perp$$

*Demostración.* Podemos escribir el producto escalar de dos vectores $c \cdot d$ como $t(c \star d)$ donde $t$ es la aplicación que lleva un vector en la suma de sus componentes. Sean $c \in$

$C, d \in D, e \in C \star D$ vectores arbitrarios. Entonces:

$$c \cdot (d \star e) = t(c \star (d \star e)) = t((c \star d) \star e) = (c \star d) \cdot e = 0$$

Como cualquier pareja de vectores de $C, D$ es ortogonal a cualquier vector del producto, tenemos el resultado que buscábamos. □

**Proposición 3.10.** *Sean $C$ y $D$ dos códigos, de dimensiones $k_C, k_D$ respectivamente y con soporte en $1 \dots n$. Sea $E = C \star D$. Entonces*

$$d_E \leq max\{1, n - k_1 - k_2 + 2\}$$

*Además si ni $C$ ni $D$ ni $(C \star D)^\perp$ es el código de repetición $Rep_q(n)$, entonces se da la igualdad si ambos son códigos GRS sobre la misma n-upla $\alpha$.*

*Demostración.* Como $C, D$ tienen soporte en $1 \dots n$, sabemos que para cualquier $d \in D$ de peso mínimo, existe una palabra $c \in C$ de forma que $c \star d \neq 0$. Esto implica que la distancia mínima $d_E$ de $E$ será más pequeña que tanto la de $C$ como la de $D$.

Aplicando el lema 3.9 con $D$ y $(E)^\perp$ tenemos que $dim(D \star E^\perp) \geq k_D + d_E - 2$.

Ahora aplicando el lema 3.10 tenemos que $k_C + dim(D \star E^\perp) \leq n$ y por tanto:

$$k_C \leq n - dim(D \star E^\perp) \leq n - k_D - d_E + 2$$

Por último despejando obtenemos el resultado buscado.

Para la particularidad de los GRS nos vale con aplicar la proposición 3.3. □

Este resultado nos dice que si los datos han sido codificados con un código GRS $C$, y queremos proteger contra un número fijo de servidores cooperantes $b$, la mejor opción para el código de las solicitudes de $D$ es otro código GRS que cumpla la cota del teorema anterior. De existir, este código sería de dimensión $k_D = n - (n - k_D) = d_{D^perp} - 1 = b$ y lograría un cPoP de $n/(n - k_C - b + 2 - 1) = n/(n - k_C - b + 1)$

**Teorema 3.11.** *Sea $C = GRS_j(\alpha, v)$ el código usado para codificar una base de datos en $n$ servidores. Entonces para cada $1 \leq b \leq n - k$ existe un código para las solicitudes $D$ tal que el protocolo de la sección anterior protege contra $b$ servidores cooperantes y con el que se consigue un cPoP de $n/(n - k_C - b + 1)$*

*Demostración.* Demostraremos este teorema construyendo un código $D$ con estas características. Este código cumplirá que $b = d_{D_\perp} - 1$ y $d_{(C \star D)} - 1 = n - k - b + 1$. Sea $D = GRS_b(\alpha, u)$ para un vector arbitrario $u \in \mathbb{F}_q^n$. Su dual es otro código GRS con $d_{D_\perp} = b - 1$ y también $C \star D = GRS_{k+b-1}(\alpha, u \star v)$, por lo que $d_{(C \star D)} = n - k - b + 1 + 1$ como queríamos y por ello, este código cumple todas las características necesarias. □

### 3.3.1. Un ejemplo en SAGE

En primer lugar crearemos un código GRS sobre el cuerpo finito de 11 elementos. Este será el código usado para codificar los archivos. Para crear el código usamos un método de la librería codes de Sage al que solo tenemos que darle el vector de puntos de evaluación, la dimensión del código y los multiplicadores.

```
In [1]: F=GF(11)

        #vector de puntos de evaluación
        alpha=vector(F,[2,4,5,3,7,8,9,10,6,1,0])

        #vector de multiplicadores
        va=vector(F,[2,4,7,2,4,4,6,4,5,7,1])

        C=codes.GeneralizedReedSolomonCode(alpha,4,va)
        C

Out[1]: [11, 4, 8] Generalized Reed-Solomon Code over GF(11)
```

Ahora creamos la base de datos sin codificar. Estará formada por dos archivos X1 y X2. Como la dimensión del código es 4 y vamos a usar 3 bandas cada uno de ellos lo podemos representar como una matriz $3 \times 4$. Posteriormente los combinamos para obtener la base de datos sin codificar

```
In [2]: X1=Matrix(F,[[2,3,5,10],[1,7,0,4],[5,8,1,1]]);  #primer archivo
        X2=Matrix(F,[[4,4,3,4],[9,10,2,3],[4,6,2,10]]); #segundo archivo
        X=X1.stack(X2);
        X

Out[2]: [ 2  3  5 10]
        [ 1  7  0  4]
        [ 5  8  1  1]
        [ 4  4  3  4]
        [ 9 10  2  3]
        [ 4  6  2 10]
```

La base de datos codificada la podemos obtener multiplicando la base de datos y la matriz generatriz del código

```
In [3]: Y=X*C.generator_matrix();
        Y

Out[3]: [ 7 10  9  3  8  7  1  9  8  8  2]
        [ 6  7  1  7  1  5  5  4  3  7  1]
        [ 0  6  1  9  8  6  9 10  7  6  5]
        [ 2  9  2  5  0  7 10  7  6  6  4]
        [ 1  3  0  1  6  5  3  3  7  3  9]
        [10  6 10  4  7  3  4  4  8  0  4]
```

Suponemos que queremos recuperar **el primer** archivo protegiéndonos contra $b = 2$ servidores cooperantes. La última sección nos dice que el código $D$ que usaremos para las solicitudes será un código GRS de dimensión 2. En este caso obtenemos que $c = d_{C\star D} - 1 = n - (k + b - 1) + 1 - 1 = 6$. Efectivamente necesitamos por tanto 3 bandas y 2 sub-solicitudes. El código que elegiremos $D$ será el siguiente:

```
In [4]: vb=vector(F,[6,6,5,1,5,9,1,1,3,5,1])      #vector de multiplicadores
        D=codes.GeneralizedReedSolomonCode(alpha,2,vb);
        D
```

```
Out[4]: [11, 2, 10] Generalized Reed-Solomon Code over GF(11)
```

Procedemos a crear ahora los vectores aleatorios de $D$ para la primera sub-solicitud. Necesitamos tantos como bandas haya en la base de datos, es decir $3 \times 2 = 6$. Los representaremos como una matriz donde cada fila es uno de los vectores aleatorios. Cada uno de los $d_j$ será la j-ésima columna de la matriz.

```
In [5]: d=matrix(F,0,11);
        for i in range(6):
            #Creamos un elemento y lo añadimos como fila a la matriz
            d=d.stack(D.random_element());
        d
```

```
Out[5]: [ 9  6  1  4  4  0  8  5  7  6  2]
        [ 0  7  6  7 10  4  5  1  7  9  8]
        [ 6  9  6  4  3  6  0  3  6  1  6]
        [ 8  5  2  2  5  4  6  3  1  7  0]
        [ 4 10  9  3  3  0  6  1  8 10  7]
        [ 8  2  1 10  7  7  7  1  9  0  6]
```

Ahora generaremos las sub-solicitudes a partir de estos vectores aleatorios. Cada una de estas estará formada por la suma de una columna de $d$ y un vector canónico en algún caso. Tenemos que ver para que servidores es necesario añadir este vector canónico. El conjunto $J$ de servidores de donde extraeremos símbolos del archivo es de tamaño $max\{c,k\} = c = 6$, por tanto serán los 6 primeros. En cada solicitud extraeremos $c = 6$ símbolos, uno por cada uno de los primeros 6 servidores. Tenemos que dividir ahora los 6 servidores en $\alpha = 3$ grupos de 2 elementos. Tenemos entonces 3 grupos que son los siguientes:

$$J_1^1 = \{1,2\}, J_1^2 = \{3,4\}, J_1^3 = \{5,6\}$$

Con los grupos ya formados podemos formar las solicitudes. La sub-solicitud que enviaremos al servidor $j$ será la suma de la columna $j$ de d y un vector con un uno en la posición correspondiente al grupo de $j$ (solo para los servidores en $J$). Para los del primer grupo el uno estará en la primera fila, a los del segundo en la segunda, y a los del tercero en la tercera.

Si ahora ponemos cada una de las sub-solicitudes como un vector fila nos queda que las sub-solicitudes son las siguientes:

```
In [6]: S1=[]
        for i in range(11):
            v=d.column(i);
            j=i//2;
```

```
        if j<3:            #Para los tres primeros grupos
            #sumamos una unidad en la fila del grupo correspondiente
           v[j]=v[j]+1;
        S1.append(v);
    S1
```

```
Out[6]: [(10, 0, 6, 8, 4, 8),
         (7, 7, 9, 5, 10, 2),
         (1, 7, 6, 2, 9, 1),
         (4, 8, 4, 2, 3, 10),
         (4, 10, 4, 5, 3, 7),
         (0, 4, 7, 4, 0, 7),
         (8, 5, 0, 6, 6, 7),
         (5, 1, 3, 3, 1, 1),
         (7, 7, 6, 1, 8, 9),
         (6, 9, 1, 7, 10, 0),
         (2, 8, 6, 0, 7, 6)]
```

Podemos realizar ahora el mismo proceso para la segunda sub-solicitud y luego decodificar las dos sub-solicitudes. Para la segunda sub-solicitud tenemos que rotar dos posiciones los elementos de cada grupo de servidores. Así los grupos que nos quedan son los siguientes:

$$J_2^1 = \{3,4\}, J_2^2 = \{5,6\}, J_2^3 = \{1,2\}$$

Realizamos el mismo proceso para nuevos vectores aleatorios d y obtenemos lo siguiente:

```
In [7]: d=matrix(F,0,11);
        for i in range(6):
            d=d.stack(D.random_element());
        d

        S2=[]
        for i in range(11):
            v=d.column(i);
            j=i//2;
            if j<3:
                v[(j+1)%3]=v[(j+1)%3]+1;
            S2.append(v);
        S2
```

```
Out[7]: [(1, 3, 8, 10, 9, 1),
         (6, 7, 2, 1, 1, 3),
         (8, 3, 2, 9, 3, 7),
         (7, 8, 0, 0, 10, 4),
         (4, 10, 7, 7, 0, 5),
         (3, 10, 7, 2, 5, 5),
         (4, 10, 7, 1, 6, 5),
         (9, 3, 1, 3, 9, 7),
```

```
(0, 5, 9, 7, 2, 8),
(7, 0, 0, 2, 9, 0),
(3, 7, 6, 5, 1, 9)]
```

Realizamos las consultas a la base de datos. Recibimos dos símbolos de cada vector, el primero la respuesta a la primera sub-solicitud y el segundo la respuesta a la segunda sub-solicitud. Agrupamos las 10 respuestas que obtenemos de cada sub-solicitud en un mismo vector, obteniendo dos, uno por cada sub-solicitud:

```
In [8]: R1=[]
        R2=[]
        for i in range(11):
            #Producto de la sub-solicitud 1 i por los datos del servidor i
            R1.append(S1[i]*Y.column(i))
            #Producto de la sub-solicitud 2 i por los datos del servidor i
            R2.append(S2[i]*Y.column(i))
        R1=vector(R1);
        R2=vector(R2);

        [R1,R2]

Out[8]: [(5, 7, 3, 3, 9, 1, 7, 8, 0, 2, 8), (9, 8, 0, 4, 1, 2, 0, 3, 0, 7, 9)]
```

Primero recuperaremos archivos de la primera sub-solicitud y posteriormente haremos lo mismo para la segunda. El primer paso es multiplicar por la matriz de control de $C \star D$. Al ser códigos GRS podemos calcular fácilmente el código resultado del producto estrella como en la proposición 4.3. Será el siguiente:

```
In [9]: star=vector(F,11);       #realizamos el producto estrella de va y vb
        for i in range(11):
            star[i]=va[i]*vb[i];

        E=codes.GeneralizedReedSolomonCode(alpha,5,star) #Creamos el código
        E

Out[9]: [11, 5, 7] Generalized Reed-Solomon Code over GF(11)
```

Ahora multiplicamos una matriz de control de $C \star D$ por el vector de respuestas:

```
In [10]: H=E.parity_check_matrix()
         s=H*R1.column()
         s

Out[10]: [8]
         [9]
```

```
[4]
[2]
[7]
[7]
```

Ahora solo tenemos que resolver el sistema lineal $H * v = s$. El vector v tendrá en las seis primeras componentes los símbolos $y_{1,1}, y_{1,2}, y_{2,3}, y_{2,4}, y_{3,5}, y_{3,6}$

```
In [11]: v1=E.parity_check_matrix().solve_right(s);
         v1

Out[11]: [ 7]
         [10]
         [ 1]
         [ 7]
         [ 8]
         [ 6]
         [ 0]
         [ 0]
         [ 0]
         [ 0]
         [ 0]
```

Realizamos el mismo proceso a continuación en la segunda sub-solicitud para obtener los símbolos que nos faltan $y_{2,1}, y_{2,2}, y_{3,3}, y_{3,4}, y_{1,5}, y_{1,6}$

```
In [12]: H=E.parity_check_matrix()
         s=H*R2.column()

         v2=E.parity_check_matrix().solve_right(s);
         v2

Out[12]: [6]
         [7]
         [1]
         [9]
         [8]
         [7]
         [0]
         [0]
         [0]
         [0]
         [0]
```

Hemos recibido de cada banda cuatro elementos por lo que podemos usar el código $C$ para recuperar los elementos del archivo sin codificar. Interpretamos las posiciones que no conocemos como borrones y hacemos la decodificación de borrones.

In [13]:
```
G=C.generator_matrix()

b=[]
b.append(vector(F,[7,10,8,7]))  #Elementos conocidos de la banda 1
b.append(vector(F,[6,7,1,7]))   #Elementos conocidos de la banda 2
b.append(vector(F,[1,9,8,6]))   #Elementos conocidos de la banda 3


Gs=[]
#Proyección de C a las posiciones conocidas de la banda 1
Gs.append(G[[0,1,2,3],[0,1,4,5]])
#Proyección de C a las posiciones conocidas de la banda 2
Gs.append(G[[0,1,2,3],[0,1,2,3]])
#Proyección de C a las posiciones conocidas de la banda 3
Gs.append(G[[0,1,2,3],[2,3,4,5]])

file = Matrix(F,0,4)

#Resolución del sistema para conseguir la palabra sin codificar
for i in range(3):
    v=Gs[i].solve_left(b[i])
    file=file.stack(v)
file
```

Out[13]:
```
[ 2  3  5 10]
[ 1  7  0  4]
[ 5  8  1  1]
```

# Capítulo 4

# PIR con servidores bizantinos y no responsivos

Hasta ahora, hemos tratado el caso en el que siempre recibimos la respuesta correcta de los servidores a nuestras solicitudes. Sin embargo, en la práctica siempre pueden ocurrir problemas y que esta no sea la situación. En primer lugar, es posible que la respuesta de un servidor se pierda por un fallo en la comunicación o porque el servidor no está disponible en ese momento. A todos estos servidores en los que, de una forma u otra, no recibimos una respuesta los llamamos servidores no responsivos. Es posible que el conjunto de servidores no responsivos varíe entre una sub-solicitud y otra ya que no sabemos cuando pueden ocurrir los errores en la comunicación.

El segundo tipo de problema que puede ocurrir es que la respuesta que reciba el usuario de un servidor no sea la respuesta correcta a la solicitud enviada. Esto puede ocurrir también por un ataque o por un error en la transmisión de los datos por ejemplo por causa del uso de un canal con ruido que modifica alguno de los símbolos de la sub-solicitud o de la respuesta. A estos servidores de los que recibimos un símbolo erróneo los llamamos servidores bizantinos. A diferencia de los no responsivos, analizando únicamente la respuesta de un servidor no podemos conocer si el servidor se esta comportando de forma bizantina o no. La única forma de detectarlos es agrupando varias respuestas de distintos servidores y viendo que no son consistentes entre ellas. Por ello, la existencia de estos servidores es más peligrosa que la de los no responsivos. Al igual que en el caso anterior, de una sub-solicitud a otra, el conjunto de servidores bizantinos puede cambiar.

En este capítulo veremos como podemos adaptar protocolos de los capítulos anteriores para conseguir recuperar archivos con privacidad incluso con la presencia de servidores tanto bizantinos como no responsivos. Crearemos un primer protocolo inicial que logre estos objetivos y más adelante realizaremos modificaciones para lograr un mayor cPoP.

## 4.1. Protocolo inicial

En esta sección explicaremos como adaptar el protocolo de las secciones anteriores para poder utilizarlo en presencia de servidores bizantinos, no responsivos o incluso ambos simultáneamente. Sin embargo, como es lógico, la presencia de estos tipos de servidores

harán que el cPoP sea mayor y en caso de existir muchos de ellos en una misma sub-solicitud, podría incluso hacer imposible la decodificación del archivo. Más información de este protocolo se puede encontrar en [10].

En esta versión descargaremos una banda del archivo por sub-solicitud. Por tanto en la solicitud j-ésima obtendremos la banda $j$ del archivo. Más adelante daremos una versión mediante la cual podemos en algunos casos descargar varias bandas por sub-solicitud. Al igual que en capítulos anteriores incluimos una tabla de referencia para las variables usadas:

| **Variable** | **Significado** |
|---|---|
| $X^i$ | Archivo $i$ sin codificar, podemos representarlo como una matriz $\alpha \times k$ |
| $\alpha$ | Número de bandas en las que está dividido cada archivo. |
| $C$ | Código MDS usado para codificar los archivos en los servidores |
| $k$ | Dimensión de $C$ |
| $n$ | Número de servidores y longitud de $C$ |
| $d$ | Distancia mínima de $C$ |
| $b$ | Número de servidores cooperantes |
| $z$ | Número de servidores bizantinos |
| $r$ | Número de servidores no responsivos |
| $G$ | Matriz Generatriz de $C$ |
| $D$ | Código que usaremos para crear la parte aleatoria de las solicitudes |
| $E$ | Código que usaremos para eliminar servidores bizantinos y no responsivos |
| $v_E$ | Único generador de $E$ |
| $C^{\star D}_{\star E}$ | Código $C \star D + C \star E$ |
| $M$ | Matriz generatriz de $C^{\star D}_{\star E}$ |
| $Y^i$ | Cada uno de los archivos codificados en los servidores, $Y^i = X^i G$ |
| $m$ | Número de archivos en la base de datos |
| $f$ | Índice del archivo deseado |
| $w_l$ | Vector columna de datos contenidos en un servidor $l$ |
| $\rho$ | Número de solicitudes necesarias para obtener $X^f$ |
| $d_j$ | Vector fila formado por las componentes j-ésimas de cada palabra aleatoria $d^{p,a}$ |
| $r_{li}$ | Respuesta del servidor $l$ a la sub-solicitud $i$ |
| $r_i$ | Agrupación de las $n$ respuestas de los servidores a la primera sub-solicitud |

Supondremos que queremos que el protocolo se efectivo para $z$ servidores bizantinos y $r$ servidores no responsivos. Sea $f$ el archivo deseado. Definiremos $C$ el código usado para codificar los datos en la base de datos de dimension $k$, y $D$ el código usado para generar la parte aleatoria de las solicitudes. Al igual que antes, tendremos que definir $m\alpha$ palabras aleatorias de $D$, $d^{p,a}p \in [m], a \in [\alpha]$. Usando estas palabras formaremos de la misma forma los $d_j$ tomando la j-ésima componente de cada palabra aleatoria. La principal diferencia viene aquí. Definiremos un tercer código $E$ de dimensión 1. Este código lo podemos representar simplemente como una palabra no nula $v_E$ del código, ya que lo genera. Gracias a este código podremos solucionar todos los problemas causados por los servidores bizantinos y no responsivos.

Sin embargo, este código $E$ no lo elegiremos de forma arbitraria. Para que el protocolo funcione necesitamos que cumpla tres condiciones:

- La intersección de los códigos $C \star D$ y $C \star E$ es nula
- $C \star E$ tiene dimensión $k$
- La distancia mínima $d_\star$ del código $C^{\star D}_{\star E} = C \star D + C \star E$ cumple la siguiente desigualdad: $d_\star - 1 \geq 2z + r$

La primera sub-solicitud que enviaremos a cada servidor $l$ será la siguiente:

$$q_{l,1} = d_l + v_{E,l} e_{\alpha(f-1)+1}$$

donde $v_{E,l}$ es la l-ésima componente de $v_E$ y $e_{\alpha(f-1)+1}$ es el vector canónico con un 1 en la posición $\alpha(f-1)+1$.

Las siguientes sub-solicitudes se calcularán de la misma forma pero tomando unos vectores aleatorios distintos, resultando en vectores $d_l$ distintos, y solicitando la banda $\alpha(f-1)+j$ para la sub-solicitud $j$.

**Proposición 4.1.** *La agrupación de las respuestas recibidas por los servidores a una misma sub-solicitud, forman una palabra de $C^{\star D}_{\star E} = C \star D + C \star E$ suponiendo que todas estas respuestas son correctas, es decir, sin considerar servidores bizantinos o no responsivos.*

*Demostración.* La respuesta de un servidor a la solicitud es el producto escalar de la solicitud por su vector de datos:

$$r_{j,l} = q_l \cdot w_l = d_l \cdot w_l + v_{E,l} e_{\alpha(f-1)+l} w_l$$

Operando de forma similar a la proposición 3.4,

$$r_j = (r_{l,1} \dots r_{l,n}) = (d_l w_l)_{l \in [n]} + (v_{E,l} y^f_{j,l})_{l \in [n]}$$

donde $y^i_{j,l}$ es el símbolo almacenado por el servidor $l$ correspondiente a la banda $j$ del archivo codificado $f$.

En la proposición 3.4 ya vimos que el primer sumando es un elemento de $C \star D$. El segundo sumando de hecho es el producto estrella de $v_e$ y la banda $j$ del archivo $f$ por lo que es una palabra de $C \star E$. Por ello la suma de ambas es una palabra de $C^{\star D}_{\star E}$ □

**Teorema 4.2.** *Podemos decodificar una banda del archivo a partir de las repuestas $r_l$ de los servidores a una sub-solicitudes, donde como máximo hay $z$ servidores bizantinos y $r$ servidores no responsivos.*

*Demostración.* Podemos agrupar las $n$ respuestas de los servidores en un vector $r = (r_1 \dots r_n)$. Usando la proposición anterior sabemos que este vector es una palabra de $C^{\star D}_{\star E}$ pero con, como máximo $z$ errores y $r$ borrones. Por la tercera condición con la que definimos $E$, la distancia mínima del código $C^{\star D}_{\star E}$ cumple que $d_\star - 1 \geq 2z + r$. Por ello podemos usar el algoritmo de decodificación de este código para corregir los errores y borrones, y así obtener la palabra correcta que deberíamos haber recibido de los servidores. Llamaremos a esta palabra $r'$.

Dado que $C \star E$ y $C \star D$ tienen intersección trivial, $C^{\star D}_{\star E}$ está generado por los generadores del primero y los del segundo. Por tanto se puede definir una matriz generatriz a partir de las matrices generatrices de ambos códigos.

$$M = G_{C^{\star D}_{\star E}} = \begin{pmatrix} G_{C \star D} \\ G_{C \star E} \end{pmatrix}$$

Esto quiere decir que $r' = (y_1, \ldots, y_{k'}, y'_1, \ldots, y'_k)M$ donde $k'$ es la dimensión de $C \star D$, ya que por definición de $E$, la dimensión de $C \star E$ es $k$. Este vector, que llamaremos $v$, lo podemos obtener mediante la resolución de un sistema lineal.

En la demostración anterior obtuvimos una descomposición de $r'$:

$$r' = (d_l w_l)_{l \in [n]} + (v_{E,l} y^f_{j,l})_{l \in [n]}$$

donde el primer sumando pertenece a $C \star D$ y el segundo a $C \star E$. Por eso,

$$r' = vM = (y_1, \ldots, y_{k'})G_{C \star D} + (y'_1, \ldots, y'_k)G_{C \star E}$$

Como ambos códigos son disjuntos sabemos que los primeros sumandos de ambas expresiones de $r'$ son iguales. Lo mismo ocurre con el segundo sumando.

$$(y'_1, \ldots, y'_k)G_{C \star E} = (y^f_{j,l} v_{E,l})_{l \in [n]} = (x^f_j G_C) \star v_E = x^f_j G_{C \star E}$$

donde $x^f_j$ es la banda $j$ del archivo $f$.

Esta última igualdad nos dice que las últimas $k$ coordenadas de $v$ se corresponden con los valores de la banda del archivo. □

**Teorema 4.3.** *El protocolo protege contra $b = d_{D_\perp} - 1$ servidores cooperantes y tiene un cPoP de $(n - r)/k$.*

*Demostración.* La prueba de la protección contra servidores cooperantes es análoga a la del teorema 3.6.

Descargamos $n - r$ símbolos ya que hay $r$ servidores que no responden y únicamente necesitamos $k$. Por tanto el cPoP es el cociente entre ambos. □

Ahora veremos como podemos ampliar este protocolo para ser capaces de descargar varias bandas en una misma sub-solicitud.

La condición que hemos impuesto sobre $E$ de que los códigos $C \star D$ y $C \star E$ tengan intersección trivial, nos dice que la siguiente desigualdad se tiene que cumplir: $n \geq k' + k + 2z + r$, donde $k$ es la dimensión de $C$, $k'$ la de $C \star D$, $z$ el número de servidores bizantinos y $r$ la cantidad de servidores no responsivos. Si se da el caso de que $n \geq k' + \mu k + 2z + r$,para $\mu > 1$ podemos ampliar este protocolo para descargar $\mu$ bandas por sub-solicitud.

En este nuevo protocolo la única diferencia es que pediremos que la dimensión de $E$ sea $\mu$ y la dimensión de $C \star E$ sea $k\mu$. El procedimiento sería exactamente el mismo que en el caso descrito de $\mu = 1$.

Es lógico plantearnos si dados los códigos $C$ y $D$ existe un código $E$ que tal que se cumplan las tres propiedades de 4.1. Y en caso de existir como podremos obtener este código. La respuesta a estas preguntas para un código arbitrario son desconocidas debido a la complejidad de los códigos resultantes del producto estrella. Sin embargo en el caso de los códigos GRS sí que podemos encontrar una respuesta a estas dos preguntas. En la siguiente sección veremos como conseguirlo y adaptaremos este protocolo para este caso de códigos concretos.

## 4.2. Particularidades con los códigos GRS

En la sección anterior hemos visto como podemos obtener PIR incluso en presencia de servidores bizantinos y no responsivos. El protocolo indicado es válido para cualquier código pero por sus buenas propiedades, es interesante ver las particularidades en los códigos GRS. Esta adaptación fue indicada por primera vez en [11], sin embargo es importante indicar que la notación usada en dicho artículo es muy diferente a la usada en esta sección.

En este caso supondremos que nuestra base de datos está codificada usando un código GRS $C = GRS_k(\alpha, v)$. Veremos qué códigos $D$ y $E$ son los óptimos para conseguir PIR contra $b$ servidores cooperantes y teniendo en cuenta que pueden haber $z$ servidores bizantinos y $r$ no responsivos. Al igual que antes supondremos que los archivos únicamente están formados por una banda y queremos obtener el archivo $f$.

En primer lugar, como el protocolo protege contra $d_{D^\perp} - 1$ servidores cooperantes, elegiremos $D$ de la misma forma que lo hicimos en la sección 3.3, $D = GRS_b(\alpha, w)$ de forma que $C \star D = GRS_{k+b-1}(\alpha, v \star w)$.

Para la elección de $E$ tenemos que tener en cuenta las condiciones 4.1.

- $E$ debe ser de dimensión uno
- $C \star E$ debe ser de dimensión $k$
- $C \star E$ debe tener intersección trivial con $C \star D$
- La distancia mínima de $C^{\star D}_{\star E}$ debe ser mayor que $2z - r$

La primera idea que podemos tener es imponer que $E$ sea también un código GRS, pero el problema es que no hay nada que nos garantice entonces que la condición de la intersección se cumpla. En su lugar, nos fijamos que el código $C^{\star D}_{\star E}$ juega un papel muy importante en el protocolo, ya que lo usamos en la corrección de errores y borrones. Por ello impondremos que éste sea un GRS.

Como hemos visto, la matriz generatriz de $C^{\star D}_{\star E}$ es de la forma:

$$\begin{pmatrix} G_{C\star D} \\ G_{C\star E} \end{pmatrix}$$

Sabemos que $C \star D$ es un código GRS de dimensión $b + k - 1$ y parámetros $\alpha, v \star w$. Por ello si queremos que $C^{\star D}_{\star E}$ sea un código GRS, tenemos que ampliar $C \star D$ a uno con los mismo parámetros pero de dimensión $b + k - 1 + k$. Esto se conseguirá consiguiendo que

la matriz generatriz de $C \star E$ coincida con las $k$ últimas filas de la matriz generatriz del código $GRS_{b+2k-1}(\alpha, v \star w)$. La única opción de $E$ para esto es un código formado por las evaluaciones de polinomios de grado $b+k$ en $\alpha$ multiplicadas por el vector $w$, o lo que es lo mismo el código generado por la palabra $v_e = (\alpha_1^{b+k-1}, \dots \alpha_n^{b+k-1}) \star w$. Veamos que efectivamente este código cumples las cuatro condiciones.

**Proposición 4.4.** *El código $E$ generado por $v_e = (\alpha_1^{b+k-1}, \dots \alpha_n^{b+k-1}) \star w$ cumple las tres primeras condiciones necesarias que debemos imponer. Además para esta elección, $C_{\star E}^{\star D} = GRS_{b+2k-1}(\alpha, v \star w)$.*

*Demostración.* En primer lugar $E$ es claramente de dimensión 1 ya que está generado por una única palabra no nula.

En segundo lugar, $C \star E$ está generado por los productos estrella de cada generador de $C$ por cada generador de $E$. Sabemos que la matriz generatriz de $C$ es de la forma 1.52. Por tanto si hacemos el producto estrella de cada una de las filas por $v_e$, el resultado es la matriz generatriz de $C \star E$:

$$\begin{pmatrix} \alpha_1^{b+k-1} & \dots & \alpha_n^{b+k-1} \\ \vdots & \ddots & \vdots \\ \alpha_1^{b+2k-2} & \dots & \alpha_n^{b+2k-2} \end{pmatrix} \cdot diag(v \star w)$$

La matriz es de Vandermonde, por lo que todas sus filas son linealmente independientes y el código es de dimensión $k$.

En tercer lugar, la matriz generatriz de $C \star D$ es, usando 1.52, la siguiente:

$$\begin{pmatrix} 1 & \dots & 1 \\ \alpha_1 & \dots & \alpha_n \\ \vdots & \ddots & \vdots \\ \alpha_1^{b+k-2} & \dots & \alpha_n^{b+k-2} \end{pmatrix} \cdot diag(v \star w)$$

Ahora si concatenamos verticalmente ambas matrices, obtenemos la matriz generatriz de $C_{\star E}^{\star D}$. Esta claramente es la matriz generatriz de $GRS_{b+2k-1}(\alpha, v \star w)$. Además esto nos dice que todas las filas son linealmente independientes por lo que la intersección de $C \star E$ y $C \star D$ es trivial.

□

La última condición más que algo que imponer, nos da una cota de hasta cuantos errores y borrones podremos corregir. Como la distancia mínima de $C_{\star E}^{\star D}$ es $n-2k-b-1+1$, el número de errores y borrones que seremos capaces de corregir es como máximo $2z+r \leq n-2k-b-1$. Por ello todos los parámetros deben cumplir la siguiente desigualdad:

$$n \geq 2k + b + 2z + r - 1$$

### 4.2.1. Un ejemplo en Sage para servidores bizantinos y no responsivos

Supongamos que tenemos dos archivos X1 y X2 sobre el cuerpo finito de 11 elementos $F$. Usamos el siguiente código lineal para codificar la base de datos de forma que cada archivo

tenga una única banda. Supondremos que deseamos obtener el primero de ellos.

```
In [1]: F=GF(11);
        alpha=vector(F,[2,3,5,1,4,8,9,6])
        v=vector(F,[2,2,3,2,6,4,2,4])
        C=codes.GeneralizedReedSolomonCode(alpha,2,v);
        print(C)

        X1=Matrix(F,[[2,3]]).transpose(); #primer archivo
        X2=Matrix(F,[[4,4]]).transpose(); #segundo archivo
        X=X1.augment(X2);
        X
```

```
[8, 2, 7] Generalized Reed-Solomon Code over GF(11)
```

```
Out[1]: [2 4]
        [3 4]
```

```
In [2]: CodedDB=X*C.generator_matrix()
        CodedDB
```

```
Out[2]: [ 9  6  0  1  9  4 10  5]
        [ 0  8  3  3  4  8  1  9]
```

Deseamos que el protocolo proteja contra $b = 2$ servidores cooperantes. El código $D$ para realizar las sub-solicitudes debe ser entonces el siguiente código

```
In [3]: D=codes.GeneralizedReedSolomonCode(alpha,2,v); D
```

```
Out[3]: [8, 2, 7] Generalized Reed-Solomon Code over GF(11)
```

Además el código $E$ debe estar generado por la siguiente palabra:

```
In [4]: E=alpha.pairwise_product(alpha).pairwise_product(alpha).pairwise_product(v)
        E
```

```
Out[4]: (5, 10, 1, 2, 10, 2, 6, 6)
```

Empezamos con la primera sub-solicitud generando una primera matriz de palabras aleatorias de $D$. Las agrupamos por columnas en la siguiente matriz

```
In [5]: u1=D.random_element().column()
        u2=D.random_element().column()
        U=u1.augment(u2)
        U
```

```
Out[5]: [ 6  8]
        [ 2  1]
        [ 2  8]
        [10  4]
        [ 5  4]
        [ 8  9]
        [ 0  3]
        [ 2  4]
```

Si ahora sumamos el vector $E$ a la columna correspondiente al archivo que deseamos (la primera), cada una de las filas formará la sub-solicitud que enviaremos a uno de los servidores.

```
In [6]: U[:,0]+=E.column()
        U
```

```
Out[6]: [ 0  8]
        [ 1  1]
        [ 3  8]
        [ 1  4]
        [ 4  4]
        [10  9]
        [ 6  3]
        [ 8  4]
```

Enviamos cada una de las filas al servidor correspondiente. Cada uno de los servidores nos responderá con el producto escalar de su sub-solicitud y sus datos. Podemos realizar estos cálculos de forma rápida a partir de la traza del producto de ambas matrices de la siguiente forma

```
In [7]: Respuesta=(U*CodedDB).numpy().diagonal()
        Respuesta=vector(F,Respuesta)
        Respuesta
```

```
Out[7]: (0, 3, 2, 2, 8, 2, 8, 10)
```

De acuerdo a la desigualdad sobre el número de errores y borrones que podemos corregir se debe cumplir $n \geq 2k + b + 2z + r - 1$, es decir, $8 \geq 4 + 2 + 2z + r - 1 \rightarrow 2z + r \leq 3$. Podemos corregir un error y un borrón o tres borrones. Usamos la clase canal para generar dichos errores.

```
In [8]: Channel=channels.ErrorErasureChannel(VectorSpace(F,8),1,1)
        Respuesta=Channel(Respuesta)

        Respuesta
```

```
Out[8]: ((0, 3, 2, 2, 0, 2, 0, 10), (0, 0, 0, 0, 1, 0, 0, 0))
```

El código $C_{\star E}^{\star D}$ tiene que ser el siguiente. Usando sus algoritmos de decodificación podemos recuperar la palabra fuente que genera la respuesta. Las últimas dos coordenadas forman el archivo que deseábamos.

```
In [9]: vv=v.pairwise_product(v)
        Star=codes.GeneralizedReedSolomonCode(alpha,5,vv)
        decoder=codes.decoders.GRSErrorErasureDecoder(Star)

        Archivo=decoder.decode_to_message(Respuesta)
        Archivo=Archivo[3:5]
        Archivo
```

```
Out[9]: (2, 4)
```

## 4.3. Versión mejorada del protocolo para servidores bizantinos y no responsivos

Concluimos la sección anterior con una desigualdad que nos daba la cantidad de errores y borrones que podemos corregir usando el anterior protocolo. Dicha desigualdad era la siguiente:

$$n \geq 2k + b + 2z + r - 1$$

Podemos entender esta desigualdad como un desglose de para qué usamos las dimensiones: $b$ para protegernos contra servidores cooperantes, $2z + r$ para corregir errores y borrones, $k$ para descargarnos $k$ símbolos y otras $k$ para ser capaces de decodificar el "ruido" aleatorio. Todo esto nos da a pensar que si tenemos una cota para todos estos parámetros, es posible que la igualdad no se dé y por lo tanto estemos "desperdiciando" dimensiones que podríamos usar para por ejemplo descargar más símbolos por sub-solicitud y por tanto lograr un mejor cPoP. A continuación veremos una adaptación del protocolo anterior que logra corregir este pequeño problema. La tabla de referencia en este caso es la siguiente:

| Variable | Significado |
| --- | --- |
| $X^i$ | Archivo $i$ sin codificar, podemos representarlo como una matriz $\alpha \times k$ |
| $\alpha$ | Número de bandas en las que está dividido cada archivo. |
| $C$ | Código MDS usado para codificar los archivos en los servidores |
| $k$ | Dimensión de $C$ |
| $n$ | Número de servidores y longitud de $C$ |
| $d$ | Distancia mínima de $C$ |
| $b$ | Número de servidores cooperantes |
| $z$ | Número de servidores bizantinos |
| $r$ | Número de servidores no responsivos |
| $c$ | $n - k - b - 2z - r + 1$ |
| $G$ | Matriz Generatriz de $C$ |
| $D$ | Código que usaremos para crear la parte aleatoria de las solicitudes |
| $\phi$ | Función $\phi(l, i) = ic - lk + k + b - 1$ |
| $E_i$ | Código usado para quitar errores y borrones en la sub-solicitud $i$ |
| $v_l^{E_i}$ | Generador de $E_i$ correspondiente a la potencia $phi(l, i)$ de $\alpha$ |
| $C_{\star E}^{\star D}$ | Código $C \star D + C \star E$ |
| $G_\star$ | Matriz generatriz de $C_{\star E}^{\star D}$ en la sub-solicitud 1 |
| $g_t^\star$ | Generador de $C_{\star E}^{\star D}$ correspondiente a la potencia $t$ de $\alpha$ |
| $Y^i$ | Cada uno de los archivos codificados en los servidores, $Y^i = X^i G$ |
| $m$ | Número de archivos en la base de datos |
| $f$ | Índice del archivo deseado |
| $w_l$ | Vector columna de datos contenidos en un servidor $l$ |
| $\rho$ | Número de solicitudes necesarias para obtener $X^f$ |
| $d_j$ | Vector fila formado por las componentes j-ésimas de cada palabra aleatoria $d^{p,a}$ |
| $r_{li}$ | Respuesta del servidor $l$ a la sub-solicitud $i$ |
| $r_i$ | Agrupación de las $n$ respuestas de los servidores a la primera sub-solicitud |

Las cantidad de símbolos que podemos descargar en este caso viene dada por la desigualdad anterior y por tanto la definimos como $c = n - k - b - 2z - r + 1$. Definiremos el número de bandas y sub-solicitudes necesarios al igual que en la sección 3.2 de la siguiente forma:

$$\alpha = \frac{mcm(c, k)}{k} \qquad \rho = \frac{mcm(c, k)}{c}$$

Supondremos que la base de datos está generada por un código $C = GRS_k(\alpha, v)$ y que deseamos obtener el archivo $f$ de entre los $m$ archivos disponibles. Usaremos un código $D = GRS_b(\alpha, w)$ y otro código $E$ para generar las solicitudes, al igual que anteriormente. El proceso es muy similar a la versión anterior, pero la gran diferencia está en la forma de elegir $E$.

Al igual que antes generamos $m \cdot \alpha$ palabras aleatorias de $D$, y a partir de ellas formamos para cada $l \in [n]$ las palabras $d_l$ tomando la l-ésima componente de cada uno de los vectores aleatorios.

La elección de $E$ dependerá de la sub-solicitud, es decir, usaremos un código $E_i$ distinto en cada sub-solicitud. En primer lugar, $dim(E_i) = \lceil ic/k \rceil$. De esta forma para la última sub-solicitud, $dim(E_i) = \rho c/k = \alpha$. La matriz generatriz $G_i$ de cada código $E_i$ será la siguiente:

$$G_i = \begin{pmatrix} \alpha_1^{\phi(dim(E_i),i)} & \dots & \alpha_n^{\phi(dim(E_i),i)} \\ \alpha_1^{\phi(dim(E_i)-1,i)} & \dots & \alpha_n^{\phi(dim(E_i)-1,i)} \\ \vdots & \ddots & \vdots \\ \alpha_1^{\phi(2,i)} & \dots & \alpha_n^{\phi(2,i)} \\ \alpha_1^{\phi(1,i)} & \dots & \alpha_n^{\phi(1,i)} \end{pmatrix} \cdot diag(w)$$

donde $\phi$ es la función $\phi(l,i) = ic - lk + k + b - 1$. Es importante ver que $\phi(l,i) \geq b$ si y solo si $l \leq dim(E_i)$. Por ello los exponentes de la matriz anterior son todos mayores que $b$. Además examinando la función $\phi$ vemos que los exponentes de dos filas contiguas distan $k$ unidades entre ellos.

Denominaremos $g_l^{E_i}$ al generador de $E_i$ correspondiente a la potencia $\phi(l,i)$. Como hemos dicho, los exponentes en cada fila son $k$ unidades más grande que en la anterior. Gracias a ello, al calcular el producto estrella con $C$, tendremos que la matriz generatriz es:

$$\begin{pmatrix} \alpha_1^{\phi(dim(E_i),i)} & \dots & \alpha_n^{\phi(dim(E_i),i)} \\ \alpha_1^{\phi(dim(E_i),i)+1} & \dots & \alpha_n^{\phi(dim(E_i),i)+1} \\ \vdots & \ddots & \vdots \\ \alpha_1^{\phi(dim(E_i),i)+k-1} & \dots & \alpha_n^{\phi(dim(E_i),i)+k-1} \\ \alpha_1^{\phi(dim(E_i-1),i)} & \dots & \alpha_n^{\phi(dim(E_i-1),i)} \\ \vdots & \ddots & \vdots \\ \alpha_1^{\phi(1,i)} & \dots & \alpha_n^{\phi(1,i)} \\ \vdots & \ddots & \vdots \\ \alpha_1^{\phi(1,i)+k-1} & \dots & \alpha_n^{\phi(1,i)+k-1} \end{pmatrix} \cdot diag(v \star w)$$

El exponente en cada fila es una unidad mayor que la anterior. Por tanto esta matriz estará formada por las potencias de $\alpha$ desde $\phi(dim(E_i),i)$ hasta $\phi(1,i) + k - 1$

Esto supone varias cosas. En primer lugar, ya no podemos garantizar que $C \star D$ y $C \star E$ tengan intersección trivial. Esto se debe a que siempre tendremos $\phi(dim(E_i),i) = ic - \lceil ic/k \rceil k + k + b - 1 \leq b + k = dim(C \star D)$ . Sin embargo pese a ello podemos obtener el código $C_{\star E_i}^{\star D}$ de la misma forma. La principal diferencia es que dado que la intersección no es trivial, no podemos indicar cada palabra de forma única como una suma de una palabra de $C \star D$ y otra de $C \star E_i$ como hacíamos antes. De hecho, nos queda $C_{\star E_i}^{\star D} = GRS_{k+\phi(1,i)}(\alpha, v \star w) = GRS_{ic+k+b-1}(\alpha, v \star w)$. Visto todo esto podemos ya decir cuáles serán las sub-solicitudes que enviaremos:

$$q_{l,i} = d_l + \sum_{a=1}^{\lceil c/k \rceil} e_{\alpha(f-1)+a} g_a^{E_i}(l)$$

donde $e_{\alpha(f-1)+l}$ es el vector canónico con un 1 en la posición $\alpha(f-1)+l$ y $g_a^{E_i}(l)$ es la componente $l$ de $g_a^{E_i}$. La composición de la solicitud es la misma que en la sección anterior pero tenemos que añadir más elementos de $E$ porque ya no es de dimensión 1.

**Teorema 4.5.** *Podemos recuperar el archivo deseado $f$ a partir de las respuestas de los servidores a las sub-solicitudes $r_{l,i}$*

*Demostración.* En primer lugar veremos como podemos recuperar los primeros $c$ símbolos en la primera sub-solicitud y cómo podemos extenderlo al resto de ellas. Con “primeros” símbolos nos referiremos a las primeras $\lfloor c/k \rfloor$ bandas y los últimos $c - \lfloor c/k \rfloor k$ símbolos de la siguiente.

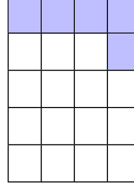

Figura 4.1: Primeros cinco símbolos de un archivo con 5 bandas de dimensión 4

De la misma forma a la sección anterior podemos observar que todas las palabras respuesta $r_i$ son una palabra de $C^{\star D}_{\star E_i}$ con a lo sumo $z$ errores y $r$ borrones. En particular, $dim(C^{\star D}_{\star E_1}) = c+t+k-1 = n-2z+t$. Esto quiere decir que podemos usar el algoritmo de decodificación para corregir los $z$ errores y $r$ borrones. Por tanto, tendremos ya el vector de respuestas corregido:

$$r_1 = (r_{11}, \ldots, r_{n1})$$

Operando igual que en la sección anterior:

$$r_1 = (d_l w_l)_{l\in[n]} + \sum_{a=1}^{\lceil c/k \rceil} (y^f_{a,l} g^{E_1}_a(l))_{l\in[n]}$$

El primer sumando es una palabra de $C \star D$ y el sumatorio es una de $C \star E_i$. Por tanto podemos expresar $r_1$ también como

$$r_1 = r_D + r_E$$

En la sección anterior sabíamos que los $b + k - 1$ primeros generadores de $C^{\star D}_{\star E_i}$ eran los generadores de $C \star D$, y el resto eran los generadores de $C \star E$. De esta forma sabíamos que las últimas coordenadas de $r'_1$ se correspondían con la expresión de $r_E$ en la base de $E$. Sin embargo, dado que en este caso la intersección de ambos no es nula, este procedimiento ya no funciona, pero podemos adaptarlo. Para ello, analizaremos los generadores de $C^{\star D}_{\star E_i}$ a partir de su matriz generatriz $G_\star$.

$$G_\star = \begin{pmatrix} 1 & \dots & 1 \\ \alpha_1 & \dots & \alpha_n \\ \vdots & \ddots & \vdots \\ \alpha_1^{b-1} & \dots & \alpha_n^{b-1} \\ \alpha_1^{b} & \dots & \alpha_n^{b} \\ \vdots & \ddots & \vdots \\ \alpha_1^{b+k-2} & \dots & \alpha_n^{b+k-2} \\ \vdots & \ddots & \vdots \\ \alpha_1^{\phi(1,i)} & \dots & \alpha_n^{\phi(1,i)} \\ \vdots & \ddots & \vdots \\ \alpha_1^{\phi(1,i)+k-1} & \dots & \alpha_n^{\phi(1,i)+k-1} \end{pmatrix} \cdot diag(v \star w)$$

Podemos denominar a cada generador de $G_\star$ como $g_t^\star$ donde $t$ es el exponente al que está elevado $\alpha$. Las primeras $b+k-1$ filas se corresponden a los generadores de $C \star D$, y las últimas $dim(E_1) \cdot k$ se corresponden a los generadores de $C \star E$. La intersección de ambos estará generada por los vectores entre las filas $\phi(dim(E_1), 1)$ y $b+k-1$. Esto nos permite separar los generadores en 3 grupos:

1. Las primeras $\phi(dim(E_1), 1)$ filas: $\{g_0^\star \dots g_{\phi(\lceil c/k \rceil),1)-1}^\star\}$ Son los generadores que solo están en $C \star D$
2. $\{g_{\phi(\lceil c/k \rceil,1)}^\star \dots g_{b+k-2}^\star\}$ Son los generadores comunes a ambos
3. Los generadores a partir de la fila $b+k$: $\{g_{b+k-1}^\star \dots g_{\phi(1,1)+k-1}^\star\}$ Son los generadores que solo están en $C \star E$

Esto nos da una descomposición única (fijando los generadores) de $r_1 = r_D + r_{DE} + r_E$ donde el primer sumando está en el primer grupo , el segundo en el segundo y el tercero en el último.

Sabemos también que por ser una palabra de $C_{\star E_i}^{\star D}$, existe un único vector $r_1'$ tal que $r_1' G_\star = r_1$ donde $G_\star$ es la matriz generatriz de dicho código. Este vector lo podemos calcular mediante la resolución de un sistema lineal.

Sea $j \leq k$ tal que $\phi(\lceil c/k \rceil, 1) + j = b + k - 1$. Sabemos que $j$ existe por cómo hemos definido el segundo grupo. Y sabemos que es menor que $k$ porque $\phi(\lceil c/k \rceil, 1) \geq b$. Aplicando la descomposición que hemos obtenido a la expresión que tenemos de $r_1$ tenemos lo siguiente:

$$r_1 = (d_l w_l)_{l \in [n]} + \sum_{a=1}^{\lceil c/k \rceil} (y_{a,l}^f g_a^{E_1}(l))_{l \in [n]} =$$

$$= (d_l w_l)_{l \in [n]} + (y_{\lceil c/k \rceil,l}^f g_{\lceil c/k \rceil}^{E_1}(l))_{l \in [n]} + \sum_{a=1}^{\lceil c/k \rceil - 1} (y_{a,l}^f g_a^{E_1}(l))_{l \in [n]} =$$

$$= (d_l w_l)_{l \in [n]} + \sum_{t=1}^{j-1} x_{\lceil c/k \rceil,t}^f g_t^C \star g_{\lceil c/k \rceil}^{E_1} + \sum_{t=j}^{k} x_{\lceil c/k \rceil,t}^f g_t^C \star g_{\lceil c/k \rceil}^{E_1} + \sum_{a=1}^{\lceil c/k \rceil - 1} \sum_{t=1}^{k} x_{a,t}^f g_t^C \star g_a^{E_1} =$$

$$= (d_l w_l)_{l\in[n]} + \sum_{t=1}^{j-1} x^f_{\lceil c/k\rceil,t} g^\star_{\phi(\lceil c/k\rceil,1)+t} + \sum_{t=j}^{k} x^f_{\lceil c/k\rceil,t} g^\star_{\phi(\lceil c/k\rceil,1)+t} + \sum_{a=1}^{\lceil c/k\rceil-1} \sum_{t=1}^{k} x^f_{a,t} g^\star_{\phi(a,1)+t}$$

En la primera igualdad hemos extraído el último término del sumatorio. En la segunda hemos expresado cada $y^f_{a,l}$ como la suma de sus coordenadas por los generadores $g^C_l$ de $C$ y además hemos separado el término que habíamos separado en dos partes, las primeras $j-1$ coordenadas y el resto. Por último hemos realizado los productos estrella entre los generadores.

El resultado son cuatro sumandos, donde el primero está generado por el primer y segundo grupo, el segundo por el segundo grupo, y los dos últimos únicamente por el tercer grupo. De hecho, las coordenadas sobre estos generadores son los $x^f_{a,t}$ que deseamos. Por tanto las coordenadas correspondientes al tercer grupo, es decir, las coordenadas de $r'_1$ a partir de la columna $b+k$ son los fragmentos del archivo que buscamos.

Examinemos ahora qué fragmentos estamos recibiendo exactamente. En primer lugar, en la última suma estamos recibiendo todos los símbolos correspondientes a las primeras $\lceil c/k\rceil - 1$ bandas (en el último sumando). Y en el tercer sumando estamos recibiendo las columnas a partir de la $j$ de la banda $\lceil c/k\rceil$. Eso hace un total de $(\lceil c/k\rceil - 1)k + k - j = \lceil c/k\rceil k - j = c$ símbolos recuperados en total.

Suponemos ahora que queremos obtener los siguientes $c$ símbolos a través de la sub-solicitud $i$ habiéndolo conseguido los primeros $c(i-1)$ en las anteriores. La respuesta que obtendremos es:

$$r_i = (d_l w_l)_{l\in[n]} + \sum_{a=1}^{\lceil ic/k\rceil} \sum_{t=1}^{k} x^f_{a,t} g^\star_{\phi(a,i)+t}$$

Veamos cómo podemos reducir este caso al caso que ya hemos visto. En primer lugar, nos fijamos que muchos de los fragmentos $x^f_{a,t}$ ya los hemos calculado en las sub-solicitudes anteriores. De hecho ya tenemos los primeros $c(i-1)$. Por ello podemos calcular cada uno de los $x^f_{a,t} g^\star_{\phi(a,i)+t}$ que conocemos y sustraerlos de la respuesta. Tras eso nos quedará un vector $r'_i$:

$$r'_i = (d_l w_l)_{l\in[n]} + \sum_{t=1}^{(i-1)j} x^f_{\lceil (i-1)c/k\rceil+1,t} g^\star_{\phi(\lceil (i-1)c/k\rceil+1,i)+t} + \sum_{a=\lceil (i-1)c/k\rceil+1}^{\lceil ic/k\rceil} \sum_{t=1}^{k} x^f_{a,t} g^\star_{\phi(a,i)+t}$$

El primer sumando se corresponde a la parte aleatoria, el tercer sumando a las bandas nuevas añadidas que no aparecían en la sub-solicitud anterior y el segundo a la banda que aparecía en la sub-solicitud anterior pero que no se recupero totalmente.

Como podemos ver, en este caso no estamos usando todo $E_i$ ya que hemos eliminado la parte correspondiente a las últimas $(i-1)c$ coordenadas. Esto quiere decir que la última coordenada no nula de $r'_i$ será $\phi(1,i) + k - 1 - (i-1)c = \phi(1,1) + k - 1$. Esto quiere decir que $r'_i$ usa exactamente los mismo generadores que $r_1$ y por ello podemos realizar el mismo procedimiento que en la primera solicitud para obtener los $c$ símbolos siguientes.

Al final de la sub-solicitud $\rho$, habremos obtenido un total de $\rho c = \alpha k$ símbolos, obteniendo el archivo completo.

$\square$

**Teorema 4.6.** *El protocolo protege contra $b = d_{D_\perp} - 1$ servidores cooperantes y tiene un cPoP de $(n-r)/c$.*

*Demostración.* La prueba de la protección contra servidores cooperantes es igual al caso de la sección 3.2.

Para el cPoP, descargamos $(n-r)$ elementos por sub-solicitud, a partir de los cuales, obtenemos realmente $c$ elementos del archivo. Por tanto el cPoP será el cociente de ambos. $\square$

### 4.3.1. Ejemplo en Sage

Sea una base de datos formada por dos archivos $X1, X2$, de 12 elementos de $\mathbb{F}_{11}$. Codificamos dicha base de datos en 11 servidores usando el siguiente código $C = GRS_4(alpha, v)$. Dividiremos cada archivo en tres bandas distintas

```
In [1]: F=GF(11)
        X1=Matrix(F,[[2,3,3,5],[6,2,6,10],[4,9,7,1]]) #primer archivo
        X2=Matrix(F,[[4,4,0,2],[4,2,4,7],[10,2,1,4]]) #segundo archivo
        X=X1.stack(X2);
        X
```

```
Out[1]: [ 2  3  3  5]
        [ 6  2  6 10]
        [ 4  9  7  1]
        [ 4  4  0  2]
        [ 4  2  4  7]
        [10  2  1  4]
```

```
In [2]: alpha=vector(F,[2,3,5,1,4,8,9,6,0,10,7])
        v=vector(F,[2,2,3,2,6,4,2,4,5,6,7])
        C=codes.GeneralizedReedSolomonCode(alpha,4,v);

        N=11; K=4;b=2;

        CodedDB=X*C.generator_matrix(); CodedDB
```

```
Out[2]: [10  5  6  4  4  2  2  3 10  4  6]
        [ 8  1  2  4  1  5  2  6  8  0  6]
        [ 6  0  2  9  9  8  1  3  9  6  2]
        [ 1  8  8  9  8  5  4  3  9 10 10]
        [ 6  8  8  1  9  7  8  0  9  5  1]
        [ 1  2  7  1  2  5  0  3  6  8  6]
```

Cada una de las columnas de esta matriz representa los datos que están almacenados en cada uno de los servidores. Suponemos que deseamos obtener el primer archivo, protegiendo contra $b = 2$ servidores cooperantes y corrigiendo un borrón y un error. Calculamos $c$, el número de símbolos que recuperaremos por sub-solicitud

```
In [3]: c=11-4-2-2*1-1+1; c
```

```
Out[3]: 3
```

Empezamos con el proceso de creación de la primera sub-solicitud. En este ejemplo únicamente realizaremos la primera y segunda sub-solicitud ya que el proceso para el resto de ellas es igual. El primer paso es generar una matriz de vectores aleatorios de $D = GRS_b(alfa, v)$.

```
In [4]: D=codes.GeneralizedReedSolomonCode(alpha,2,v)

        U=Matrix(F,11,0)
        for i in range(6):
            U=U.augment(D.random_element())
        U
```

```
Out[4]: [ 6  9  3 10 10  3]
        [ 0  2 10  8  0  9]
        [ 4  4  3  6  3  4]
        [ 1  5  7  1  9  8]
        [ 4  7  7  7  3  1]
        [ 6  0  2  7 10  1]
        [ 8  4  8  7  6  1]
        [ 8  6  7  4  6 10]
        [ 1  8  0  2  9  5]
        [ 6  2  1  4 10 10]
        [ 4  8  1  0  3  0]
```

El siguiente paso es calcular el código $E_i$ y sumar sus generadores a las columnas correspondientes. Para la primera sub-solicitud, $dim(E_1) = \lceil c/k \rceil$

```
In [5]: dim=ceil(c/K); dim
```

```
Out[5]: 1
```

$E_1$ estará generado únicamente por un vector que será formado por las potencias de *alpha* multiplicado por $v$. La potencia a la que tenemos que elevar *alpha* viene dada por la función $\phi$

```
In [6]: pot=c-K+K+b-1; print(pot)

        E1=alpha.pairwise_product(alpha).pairwise_product(alpha).
           pairwise_product(alpha).pairwise_product(v)
        E1
```

```
4
```

```
Out[6]: (10, 8, 5, 2, 7, 5, 10, 3, 0, 6, 10)
```

Se lo sumamos a la primera columna y con esto ya terminaríamos el cálculo de la primera sub-solicitud.

```
In [7]: U[:,0]+=E1.column(); U
```

```
Out[7]: [ 5  9  3 10 10  3]
        [ 8  2 10  8  0  9]
        [ 9  4  3  6  3  4]
        [ 3  5  7  1  9  8]
        [ 0  7  7  7  3  1]
        [ 0  0  2  7 10  1]
        [ 7  4  8  7  6  1]
        [ 0  6  7  4  6 10]
        [ 1  8  0  2  9  5]
        [ 1  2  1  4 10 10]
        [ 3  8  1  0  3  0]
```

Simulamos las respuestas de los servidores de la misma forma que hicimos en el ejemplo anterior y creamos los $z = 1$ errores y $r = 1$ borrones correspondientes.

```
In [8]: Respuesta=(U*CodedDB).numpy().diagonal()
        Respuesta=vector(F,Respuesta)
        Respuesta
```

```
Out[8]: (4, 3, 3, 0, 1, 5, 7, 0, 5, 4, 5)
```

```
In [9]: Channel=channels.ErrorErasureChannel(VectorSpace(F,N),1,1)
        Respuesta=Channel(Respuesta)

        Respuesta
```

```
Out[9]: ((4, 3, 3, 0, 1, 0, 7, 0, 5, 4, 9), (0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0))
```

Al tratarse de la primera sub-solicitud no necesitamos realizar ninguna sustracción. Nos vale con calcular el código $C_{\star E1}^{\star D}$ y usar los métodos de decodificación del mismo.

```
In [10]: Star=codes.GeneralizedReedSolomonCode(alpha,K+b+c-1,v.pairwise_product(v))
         decoder=codes.decoders.GRSErrorErasureDecoder(Star)

         word=decoder.decode_to_message(Respuesta)
         word
```

```
Out[10]: (9, 4, 5, 2, 2, 3, 3, 5)
```

Los $c = 3$ últimos símbolos de *word* son los tres primeros fragmentos del archivo. Por tanto hemos decodificado la siguiente parte del archivo:

```
In [11]: Archivo=Matrix(F,[[0,3,3,5],[0,0,0,0],[0,0,0,0]])
         Archivo
```

```
Out[11]: [0 3 3 5]
         [0 0 0 0]
         [0 0 0 0]
```

Procedemos de forma similar con las segunda sub-solicitud, empezando a generar la matriz $U$.

```
In [12]: D=codes.GeneralizedReedSolomonCode(alpha,2,v)

         U=Matrix(F,11,0)
         for i in range(6):
             U=U.augment(D.random_element())
         U
```

```
Out[12]: [ 4  4  1  3  4  6]
         [10  3  4  2  7  6]
         [ 0  7  4  0  3  9]
         [ 9  5  9  4  1  6]
         [ 4  6 10  3  8  7]
         [ 3  7  5  5  0  1]
         [ 2  8  0  7  3  6]
         [ 1  0  4  9 10  1]
         [ 2  4  4  7  6  4]
         [ 2 10  9  7  7  7]
         [ 9  2  1  4  6 10]
```

La dimensión de $E2$ en este caso será mayor, ya que:

```
In [13]: dim=ceil(2*c/K); dim
```

```
Out[13]: 2
```

$E2$ lo generarán dos vectores, ambos una potencia de los *alpha* multiplicado por $v$. Estas potencias son:

```
In [14]: pot1=2*c-K+K+b-1; print(pot1)
         pot2=2*c-2*K+K+b-1; print(pot2)

         E21=alpha.pairwise_product(alpha).pairwise_product(alpha)
         E21=E21.pairwise_product(E21).pairwise_product(alpha).
            pairwise_product(v)
         print(E21)

         E22=alpha.pairwise_product(alpha).pairwise_product(alpha).
            pairwise_product(v)
         print(E22)
```

```
7
3
(3, 7, 9, 2, 8, 8, 8, 10, 0, 5, 9)
(5, 10, 1, 2, 10, 2, 6, 6, 0, 5, 3)
```

El primero de ellos se lo tenemos que sumar a la primera columna de $U$ y el segundo a la segunda

```
In [15]: U[:,0]+=E21.column();
         U[:,1]+=E22.column();
         U
```

```
Out[15]: [ 7  9  1  3  4  6]
         [ 6  2  4  2  7  6]
         [ 9  8  4  0  3  9]
         [ 0  7  9  4  1  6]
         [ 1  5 10  3  8  7]
         [ 0  9  5  5  0  1]
         [10  3  0  7  3  6]
         [ 0  6  4  9 10  1]
         [ 2  4  4  7  6  4]
         [ 7  4  9  7  7  7]
         [ 7  5  1  4  6 10]
```

Calculamos las respuestas y generamos los borrones y errores

```
In [16]: Respuesta=(U*CodedDB).numpy().diagonal()
         Respuesta=vector(F,Respuesta)
         Respuesta=Channel(Respuesta)

         Respuesta
```

```
Out[16]: ((5, 6, 0, 9, 0, 5, 0, 1, 9, 3, 4), (0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0))
```

En este caso, dado que ya no estamos en la primera sub-solicitud debemos restar los símbolos que ya hemos obtenido multiplicados por el generador correspondiente.

El código $C^{\star D}_{\star E2}$ ahora está generado por las potencias de *alpha* hasta grado 10. Dado que *Star* lo forman las potencias hasta grado 7 debemos eliminar las 3 más altas, que son las correspondientes a los fragmentos que ya tenemos.

Para calcular el de potencia 10 podemos multiplicar E1 y E2 ya que el primero es la potencia 7 y el segundo la potencia 3.

```
In [17]: R=Respuesta[0]

         Gen=E21.pairwise_product(E22);
         R-=Gen*Archivo[0,3]
```

Realizamos el mismo proceso con los otros dos elementos reduciendo en uno la potencia de *alpha*

```
In [18]: Gen=E21.pairwise_product(alpha).pairwise_product(alpha).
            pairwise_product(v)
         R-=Gen*Archivo[0,2]

         Gen=E21.pairwise_product(alpha).pairwise_product(v)
         R-=Gen*Archivo[0,1]
```

El resultado es ahora una palabra de *Star* y podemos usar el código para recuperar otros *c* fragmentos (los c últimos).

```
In [19]: decoder.decode_to_message([R,Respuesta[1]])
```

```
Out[19]: (3, 4, 10, 2, 1, 6, 10, 2)
```

Guardándolo en la matriz que estamos rellenando tenemos ya decodificado del archivo lo siguiente:

```
In [20]: Archivo[0,0]=2
         Archivo[1,3]=10
         Archivo[1,2]=6
         Archivo
```

```
Out[20]: [ 2  3  3  5]
         [ 0  0  6 10]
         [ 0  0  0  0]
```

# Apéndice A

# Implementación de los protocolos PIR

En este apéndice se mostrará la implementación de de los diferentes códigos PIR descritos en este documento, además de la implementación de una serie de métodos útiles en la simulación de estos protocolos. El lenguaje usado para la implementación de estos métodos es Sage [12].

## A.1. Programas generales

### A.1.1. Codificación de archivos en base de datos

```
def PIREncodeToServers(f, alpha, C, systematic=false):

    """
        This function encodes a list of files into a distributed database
        using some code C.

        The inputs required for this function are the following:
            f: Matrix containing all files to be encoded. Each file is a
               column of f
            alpha: Number of spripes in which we want to divide each file
            C: linear code that will be used to encode the files into the
               database
            systematic: boolean indicating wether to use the systematic
               generator matrix in the enconding or not.
               If this parameter is not specified it will be asumed to false

        The output of this program will be a matrix representing the
           distributed database. Each column will be the data contained
           by one of the servers
        """
```

```
K = C.dimension();
N = C.length();
M = f.ncols();
size = f.nrows();
F = C.base_field();


#Add zeros at the eand of each file until the size can be
#divided by alpha*k

symbols = ceil(size/(alpha*K));  # number of symbols in Fq that
#will be converted into a single element of a field extension
zeros = matrix(F, symbols*alpha*K-size, M);  #adding zeros
f=f.stack(zeros);

#Divide each file into alpha stripes and k columns.
#Then concatenate verticaly all of them.

G=GF(F.order()^symbols);
Ggen=G.gen();

V,m1,m2=G.vector_space(F,map=true);

File = Matrix(G,alpha,K);

DB = Matrix(0,K);

for m in range(M):                  #each file
    for a in  range(alpha):         #each row of the new matrix
        for k in range(K):          #each colum of the new matrix
            index=a*K*symbols+k*symbols;
            v=f[range(index,index+symbols),m];
            v=vector(v);
            File[a,k]=m1(v);

    DB=DB.stack(File);
    File=Matrix(G,alpha, K);



#Use the code to encode the files into the database
if systematic:
    CodedDB=DB*C.systematic_generator_matrix();
else:
    CodedDB=DB*C.generator_matrix();
return CodedDB;
```

### A.1.2. Cálculo de las respuestas por parte del servidor

```
def PIRQueryToServer(CodedDB, query, servers=-1):

    """
    This function simulates the process of querying a distributed
       database.

    The inputs required for this function are the following:
        CodedDB: Matrix representing the distributed database
        query: List of query matrices sent to the database. Each one
           of them will be adresse to one server
        servers: indices of the servers being adressed by the queries.
           First query matrix in parameter 'query' is
           suposed to be adressed at server in first position of
           parameter 'servers' When not specified, it will
           be asumed that all servers are queried

    The output of this program will be a list of column vectors representing
       the anwers from the database. Each one of them will correspond to the
       response provided by one server.
    """

    response=[];
    N=len(query);

    if servers==-1:
        servers=range(N)

    for n in servers:
        data=CodedDB.column(n).column();
        response.append(query[n]*data);
    return response;
```

### A.1.3. Cálculo de las respuestas con ruido

```
def PIRQueryToServerNoise(CodedDB,Query,Z,R,servers=-1):

    """
    This function simulates the process of querying a distributed
       database in a channel with noise.

    The inputs required for this function are the following:
        CodedDB: Matrix representing the distributed database
        Query: List of query matrices sent to the database. Each
           one of them will be adresse to one server
        Z: Number of errors that the channel will create
        R: Number of erasures that the channel will create
```

```
    servers: indices of the servers being adressed by the queries.
        First query matrix in parameter 'query' is suposed
        to be adressed at server in first position of parameter 'servers'
        When not specified, it will be asumed that all servers are queried

The output of this program will two matrices of the same size.
    'Responses' will contain the response from
    all the servers. Each column will be the response from one
    of the servers. 'Erasures' will be a 0-1 matrix.
    A 1 in certain position means that that same position in
    'Responses' is to be interpreted as an erasure.
"""

Response=PIRQueryToServer(CodedDB,Query,servers);

rho=Response[0].nrows()
N=len(Response)
F=Response[0][0,0].parent()

Grouped=Matrix(F,rho,0)
for i in range(N):
   Grouped=Grouped.augment(Response[i])

Erasures=Matrix(GF(2),0,N)

Channel=channels.ErrorErasureChannel(VectorSpace(F,N),Z,R)

   for i in range(rho):
      x=Channel(Grouped.row(i))
      Grouped[i,:]=x[0]
      Erasures=Erasures.stack(x[1])

Response=[]
for i in range(N):
    Response.append(Grouped[:,i])


return [Response,Erasures]
```

### A.1.4. Recuperación del archivo original a partir del archivo en forma matricial

```
def DecodeToFile(File,F):

    """
    This function decodes a file in a matrix form to the original
       form of the file
```

```
    The inputs required for this function are the following:
        File: File in matrix form required to decode
        F: Field in which the original file's is encoded

    The output of this program will be the original file in a vector form.
    """

    FF=File[0,0].parent()

    V,m1,m2=FF.vector_space(F,map=true);
    file=[];

    for i in range(File.nrows()):
        for j in range(File.ncols()):
            x=File[i,j];
            file=file+list(m2(x))


    return file
```

### A.1.5. Producto estrella de dos códigos

```
def StarProduct(C,D):

    """
    This function computes the star product of two codes

    The inputs required for this function are the following:
        C: First code to make product
        D: Second product of make product

    The output of this program will be the linear code result of making
        the star product of both codes
    """

    F=C.base_field()
    N=C.length()
    n=D.length()
    K=C.dimension()
    k=D.dimension()

    if F!=D.base_field() or N!=n:
        raise ValueError('Codes dont have the same parameters so
           star product cant be computed')

    gensC=C.generator_matrix()
    gensD=D.generator_matrix()
    G=Matrix(F,0,n)
```

```
    for i in range(k):
        for j in range(K):
            G=G.stack(Matrix(F,gensD.row(i).pairwise_product(gensC.row(j))))

    return codes.LinearCode(G)
```

## A.2. Protocolos del capítulo 2

### A.2.1. Creaciones de matrices solicitud

```
def PIRQueryMatrices(C, alpha, M, b, f, eff=false):

    """
    This function creates the query matrices corresponding to the
       PIR protocols described in chapter 3

    The inputs required for this function are the following:
        C: Linear code used to encode the distributed database to
           be queried
        alpha: Number of stripes in which each file is divided
        M: Number of files in the distributed database
        b: Number of colluding servers that we want to be protected of.
        f: Index of the file we want to recover (starting with 1)
        eff: Boolean indicating if we want to use the most efficient
           version of the protocol
            in the case b>1. If not specified, false will be asumed.

    The output of this program will be a list of query matrices to be sent
       to the database.
    """



    ############### Case b=1 ############################

    def primera(C,alpha,M,f):

        N=C.length();
        K=C.dimension();
        F=C.base_field();
        x=F.gen();
        d=N-K+1;

        if alpha!=N-K:
            print("Number of stripes does not match the required number")
```

```
        return -1;

    queries=[]

    U=random_matrix(F, K, M*alpha);

    beta=alpha//K;
    r=alpha%K;
    E=Matrix(F,K,(f-1)*alpha)
        .augment(identity_matrix(F,r).stack(Matrix(F,K-r,r)))
        .augment(Matrix(F,K,beta*K+(M-f)*alpha));

    permutation=SymmetricGroup(K).gens()[0].inverse();

    for n in range(K):      #Create E matrices for the first servers
        queries.append(U+E);
        E.permute_rows(permutation);


    for s in range(beta):
        E=Matrix(F,K,(f-1)*alpha+r+s*K)
            .augment(identity_matrix(F,K))
            .augment(Matrix(F,K,(beta-s-1)*K+(M-f)*alpha));
        M=U+E;
        for j in range(K):
            queries.append(M);

    for n in srange(N-r,N):
        queries.append(U);
    return queries;


      ################## Case b<d and not most efficient ###########

def segunda(C,alpha,M,f):

    N=C.length();
    K=C.dimension();
    F=C.base_field();
    x=F.gen();
    d=N-K+1;

    if alpha!=1:
        print("Number of stripes does not match the required number")
        return -1;

    eliminate=srange(b+K,N)
    C=codes.PuncturedCode(C,eliminate)
    N=C.length();
```

```
    d=d-len(eliminate);

    queries=[]

    H=C.parity_check_matrix()
    for i in range(b+K):
        queries.append(zero_matrix(F,K,M))
    for i in range(K):                  #Each subquery
        U=random_matrix(F, M, d-1);     #Random matrix
        Q=U*H                           #Create words in dual code
        Q[f,i]=Q[f,i]+1                 #Add 1 in the correct position

        for j in range(N):
            queries[j][i,:]=Q[:,j].transpose()
    return queries;

       ################# Case b<d and most efficient ###############

def tercera(C,alpha,M,f):

    N=C.length();
    K=C.dimension();
    F=C.base_field();
    x=F.gen();
    d=C.minimum_distance();

    if alpha!=1:
       print("Number of stripes does not match the required number")
        return -1;

    delta=floor((N-b)/K)

    eliminate=srange(b+delta*K,N)
    #Puncture code so that only b+delta*k columns remain
    C=codes.PuncturedCode(C,eliminate)
    N=C.length();                         #Adapt parameters to new code
    d=d-len(eliminate);

    comunes=srange(N-b,N)                 #Common servers to each group
    todos=range(N)

    ParityMatrices=[]

    for i in range(delta):            #For each group
        group=srange(K*i,K*(i+1))+comunes
        quitar=[item for item in todos if item not in group]
        #Get code generated by these servers
        D=codes.PuncturedCode(C,quitar);
        #Get inicial part of the parity check matrix from generator matrix
```

```
        H=-D.systematic_generator_matrix()[:,K:K+b].transpose();
        ParityMatrices.append(H);

    queries=[]
    #Inicialize subqueries
    for i in range(N):
        queries.append(zero_matrix(F,ceil(K/delta),M))


    for i in range(ceil(K/delta)):
        #Create random matrix
        U=random_matrix(F, M, b);
        #Part corresponding to common servers is the identity matrix
        for j in range(b):
           queries[N-b+j][i,:]=U[:,j].transpose()
        #Rest is calculate like in case 2
        for j in range(delta):
           Q=U*ParityMatrices[j]
           Q[f,i]=Q[f,i]+1
           #Each column is one of the subqueries
           for k in range(K):
                queries[j*K+k][i,:]=Q[:,k].transpose()
    return queries


      ############### LLamadas a las funciones ############################

queries=[];
d=C.minimum_distance()

if b>=d:
    print("PIR cannot be achieved with that many colluding servers")
    return -1

elif b==1:
    queries=primera(C,alpha,M,f)

elif  not eff :
    queries=segunda(C,alpha,M,f)

else:
    queries=tercera(C,alpha,M,f)

return queries;
```

### A.2.2. Recuperación del archivo a partir de las respuestas

```
def PIRDecodeFromResponses(C,alpha,b,R,eff=false):

        """
        This function recovers the desired file from the responses of the
            databse to the queries create by
            PIRQueryMatrices() program

        The inputs required for this function are the following:
            C: Linear code used to encode the distributed database to
                be queried
            alpha: Number of stripes in which each file is divided
            b: Number of colluding servers that we want to be protected of.
            R: List of responses from the servers of the database
            eff: Boolean indicating if we want to use the most efficient
                version of the protocol
                in the case b>1. If not specified, false will be asumed.

        The output of this program will be the file that we wanted in its
           original form
        """




        ############## Case b=1 ####################

        def primera(C,alpha,R):

            N=C.length();
            K=C.dimension();
            F=C.base_field();
            FF=R[0][0,0].parent();
            x=F.gen();
            y=FF.gen();

            beta=alpha//K;
            r=alpha%K;
            rho=R[0].nrows();

            if alpha!=N-K:
                print("Number of stripes does not match the required number")
                return -1;

            File=Matrix(FF,alpha,K);

            M=Matrix(F,rho,0);
```

```
    for i in range(N):
        M=M.augment(R[i]);

    G=C.systematic_generator_matrix();

    GG=Matrix(FF,G);


   for i in range(rho):

        S=GG;
        T=Matrix(FF,r,N);

        for j in range(r):
            T[j,(K-j+i)%K]=1;
        S=S.stack(T);

        for j in range(beta):
            a1=Matrix(FF,K,(j+1)*K);
            a2=Matrix(GG.submatrix(0,(j+1)*K,K,(j+2)*K))
            a3=Matrix(FF,K,K*(beta-1-j))
            T=a1.augment(a2).augment(a3);
            S=S.stack(T);

        s=S.solve_left(M.row(i));

        for j in range(r):
            File[j,(K+i-j)%K]=s[K+j];

        for j in range(beta):
            File[r+j*K+i,0:K]=s[K+r+j*K:r+(j+1)*K];

    return File



################## Case b<d and not most efficient ###########



def segunda(C,alpha,R):

    FF=R[0][0,0].parent();
    K=C.dimension();

    File=Matrix(FF,alpha,K);

    if alpha!=1:
```

```
        print("Number of stripes does not match the required number")
        return -1;

    for i in range(len(R)):
        #Result is the sum of the values received

        File=File+R[i].transpose()

    return File



################## Case b<d and most efficient ###############




def tercera(C,alpha,R):

    N=C.length();
    K=C.dimension();
    F=C.base_field();
    FF=R[0][0,0].parent();
    x=F.gen();
    y=FF.gen();

    delta=floor((N-b)/K)

    File=Matrix(FF,alpha,K);

    eliminate=srange(b+delta*K,N)
    #Puncture code in the last positions
    D=codes.PuncturedCode(C,eliminate)
    N=D.length();

    perGroup=ceil(K/delta)
    Symbols=vector(FF,delta*perGroup);
    posicionesTodas=[]

    for i in range(delta):
        posiciones=srange(i*K,i*K+perGroup)
        posicionesTodas=posicionesTodas+posiciones
        #Add responses received from servers of a same group
        for j in range(K):
            Symbols[i*perGroup:(i+1)*perGroup]=
               Symbols[i*perGroup:(i+1)*perGroup]+R[i*K+j]
               .transpose().row(0)
        for j in range(b):
            Symbols[i*perGroup:(i+1)*perGroup]=
```

```
                Symbols[i*perGroup:(i+1)*perGroup]+R[N-1-j]
                .transpose().row(0)

        #Recover the word that generates the codeword in Symbols

        quitar=[item for item in range(N) if item not in posicionesTodas]

        C=codes.LinearCode(Matrix(FF,
           C.systematic_generator_matrix()[:,posicionesTodas]))
        File=Matrix(C.unencode(Symbols));
        return File


    ################# Reconstruct original file ################

    F=C.base_field();

    if b==1:
        File=primera(C,alpha,R)
    elif not eff:
        File=segunda(C,alpha,R)
    else:
        File=tercera(C,alpha,R)

    file=DecodeToFile(File,F)


    return file
```

## A.3. Protocolo para códigos generales

### A.3.1. Creación de matrices solicitud

```
def PIRStarQueryMatrices(C,D,alfa,M,f,d=-1):

    """
    This function creates the query matrices corresponding to the first
        PIR protocol described in
        chapter 4

    The inputs required for this function are the following:
        C: Linear code used to encode the distributed database to be
            queried
        D: Linear code used to create the queries
        alpha: Number of stripes in which each file is divided
        M: Number of files in the distributed database
        f: Index of the file we want to recover (starting with 1)
```

```
    d: Minimum distance of the star product between C and D. If not
        specified it wil be calculated

The output of this program will be a list of query matrices to be
    sent to the database.
"""

N=C.length()
K=C.dimension()
F=C.base_field()


if d==-1:
    Star=StarProduct(C,D)
    d=Star.minimum_distance()
c=d-1

alpha=lcm(c,K)/K
rho=lcm(c,K)/c
g=c/alfa
size=max(c,K)

if alpha!=alfa:
        print("Number of stripes does not match the required number")
        return -1;

queries=[]
for i in range(N):
    queries.append(Matrix(F,rho,alfa*M))

for r in range(rho):
    #Generate random codewords
    U=Matrix(F,N,0)
    for i in range(M):
        for j in range(alfa):
            U=U.augment(D.random_element().column())
    for i in range(N):
        queries[i][r,:]=U[i,:]
        pos=floor(((i-r*g)%size)/g)
        if i<size and pos<alfa:
            queries[i][r,(f-1)*alfa+pos]+=1

return queries
```

### A.3.2. Recuperación del archivo a partir de las respuestas

```
def PIRStarDecodeFromResponses(C,Star,alfa,R,d=-1):

       """
       This function recovers the desired file from the responses
           of the database to the queries create by
           PIRStarQueryMatrices() program

       The inputs required for this function are the following:
           C: Linear code used to encode the distributed database to be
               queried
           Star: Star product of codes C and D used to create queries
           alpha: Number of stripes in which each file is divided
           R: List of responses from the servers of the database
           d: Minimum distance of the star product between C and D. If
               not specified it wil be calculated

       The output of this program will be the file that we wanted in
            its original form
       """

       N=C.length()
       K=C.dimension()
       F=C.base_field()
       FF=R[0][0,0].parent()

       if d==-1:
           d=Star.minimum_distance()

       c=d-1

       alfa=lcm(c,K)/K
       rho=lcm(c,K)/c
       g=c/alpha
       size=max(c,K)

       if alpha!=alfa:
               print("Number of stripes does not match the required number")
               return -1;

        H=Star.parity_check_matrix()
       Response=Matrix(FF,rho,0)
       for i in range(N):
           Response=Response.augment(R[i])

       File=Matrix(FF,0,K)
       Obtenidos=Matrix(FF,alfa,size)
```

```
#Lower bound of the servers from wich we receive symbols in
#first subquery
downlimit=0

#Upper bound of the servers from wich we receive symbols in
#first subquery
uplimit=c

#Recover fragments and place them in their position in the file

for i in range(rho):
    if downlimit < uplimit:
        rango=range(downlimit,uplimit);
    else:
        rango=range(0,uplimit)+range(downlimit,size)
    Hi=H[:,rango]
    x=Hi.solve_right(H*Response.row(i).column())

    if downlimit < uplimit:
        posicionR=0
    else:
        posicionR=uplimit

    posicionO=downlimit
    for a in range(alfa):
        for j in range(g):
            Obtenidos[a,posicionO]=x[posicionR,0]
            posicionR=(posicionR+1)%c
            posicionO=(posicionO+1)%size

    downlimit=(downlimit+g)%size
    uplimit=(uplimit+g)%size

#Decode each stripe

#Lower bound of the servers from which we receive symbols in
#the first stripe
downlimit=0

#Upper bound of the servers from which we receive symbols in
#the first stripe
uplimit=K

for i in range(alfa):
    if downlimit<uplimit:
        rango=range(downlimit,uplimit)
    else:
        rango=range(0,uplimit)+range(downlimit,size)
```

```
        simbolos=Obtenidos[i,rango]
        G=Matrix(FF,C.generator_matrix()[:,rango])
        x=G.solve_left(simbolos)
        File=File.stack(x)
        downlimit=(downlimit+g)%size
        uplimit=(uplimit+g)%size


    ################# Reconstruct original file ################


    file=DecodeToFile(File,F)


    return file
```

### A.3.3. Obtención de los códigos D y Star para el caso GRS

```
def GetGRSQueryCode(C,b):

    """
    This function provides the optimal codes to use in functions
        PIRStarQueryMatrices() and PIRStarDecodeFromResponses()
        depending on the number of colluding server to protect against

    The inputs required for this function are the following:
        C: Linear code used to encode the distributed database to be
           queried
        b: Number of colluding serverst that we want to protect against

    The output of this program will be the query code D to use to compute
        queries and the star product of C and D to be used in the
        decoding of the responses
    """

    N=C.length()
    K=C.dimension()
    if b>N-K:
        raise ValueError('PIR cannot be achieve against this amount of
           colluding servers')
    D=codes.GeneralizedReedSolomonCode(C.evaluation_points(),b,
       vector(C.base_field(),ones_matrix(1,N).row(0)))
    Star=codes.GeneralizedReedSolomonCode(C.evaluation_points(),K+b-1,
       C.column_multipliers())
    return(D,Star)
```

## A.4. Protocolo inicial para servidores bizantinos y no responsivos

### A.4.1. Creación de matrices solicitud

```
def PIRBizantineQueryMatrices(C,b,alfa,M,f):

        """
        This function creates the query matrices corresponding to the
            first PIR protocol described in chapter 4 against byzantine
            and non responsive servers

        The inputs required for this function are the following:
            C: Linear code used to encode the distributed database to be
                queried. Has to be a GRS code
            b: Number of colluding servers that we want to protect against
            alpha: Number of stripes in which each file is divided
            M: Number of files in the distributed database
            f: Index of the file we want to recover (starting with 1)

        The output of this program will be a list of query matrices to be sent
            to the database.
        """

        N=C.length()
        K=C.dimension()
        rho=alfa
        F=C.base_field()

        alphaa=C.evaluation_points()
        D=codes.GeneralizedReedSolomonCode(alpha,b,C.column_multipliers())

        E=vector(F,N)
        for i in range(N):
            E[i]=alpha[i]^(b+K-1)*C.column_multipliers()[i]

        queries=[]
        for i in range(N):                              #Inicialize suqueries
            queries.append(Matrix(F,rho,alfa*M))
        for r in range(rho):
            #Generate random words
            U=Matrix(F,N,0)
            for i in range(M):
                for j in range(alfa):
                    U=U.augment(D.random_element().column())

            for i in range(N):
                queries[i][r,:]=U[i,:]                      #Random part
```

```
            queries[i][r,(f-1)*alfa+r]+=E[i]  #Part corresponding to E

    return queries
```

### A.4.2. Recuperación del archvio a partir de las respuestas

```
def PIRBizantineDecodeFromResponse(C,b,alfa,R,Erasures):

    """
    This function recovers the desired file from the responses of the
        databse to the queries create by PIRBizantineQueryMatrices()
        program

    The inputs required for this function are the following:
        C: Linear code used to encode the distributed database to be
            queried
        b: Number of colluding servers that we want to protect against
        alpha: Number of stripes in which each file is divided
        R: Matrix of responses from the servers of the database. Each
            column is the response from one server
        Erasures: Matrix of the same size as 'R' containing elements
            of the two element field. Positions with a one means that
            an erasure has been received in that position

    The output of this program will be the file that we wanted in its
        original form
    """

    N=C.length()
    K=C.dimension()
    k=K+b-1

    F=C.base_field()
    FF=R[0][0,0].parent()

    evalp=vector(FF,C.evaluation_points())
    mult=C.column_multipliers()
    Star=codes.GeneralizedReedSolomonCode(evalp,2*K+b-1,
       mult.pairwise_product(mult))
    decoder=codes.decoders.GRSErrorErasureDecoder(Star)


    rho=alfa

    File=Matrix(FF,alfa,K)

    Response=Matrix(FF,rho,0)
    for i in range(N):
```

```
        Response=Response.augment(R[i])

    for i in range(rho):

        word=decoder.decode_to_message([Response.row(i), Erasures.row(i)])
        File[i,:]=word[k:len(word)]

    file=DecodeToFile(File,F)


    return file
```

## A.5. Protocolo para servidores bizantinos y no responsivos mejorado

### A.5.1. Creación de las matrices solicitud

```
def PIRBetterBizantineQueryMatrices(C,b,z,r,alfa,M,f):

        """
      This function creates the query matrices corresponding to the second
          PIR protocol described in chapter 4 against byzantine and non
          responsive servers

      The inputs required for this function are the following:
          C: Linear code used to encode the distributed database to be
              queried. Has to be a GRS code
          b: Number of colluding servers that we want to protect against
          z: Number of byzantine servers that we want to protect against
          r: Number of non-responsive servers that we want to protect against
          alpha: Number of stripes in which each file is divided
          M: Number of files in the distributed database
          f: Index of the file we want to recover (starting with 1)

      The output of this program will be a list of query matrices to be sent
          to the database.
      """

      N=C.length()
      K=C.dimension()
      c=N-K-b-2*z-r+1
      if c<1:
          raise ValueError('No se puede realizar PIR para esos parámetos')

      mcm=lcm(c,K)
      a=mcm/K
      rho=mcm/c
```

```
    if a!=alfa:
        print("Number of stripes does not match the required number")
        return -1;

    alpha=C.evaluation_points()


    mult=C.column_multipliers()
    D=codes.GeneralizedReedSolomonCode(alpha,b,mult)

    queries=[]
    for i in range(N):                                #Initialize queries
        queries.append(Matrix(F,rho,alfa*M))

    G=Matrix(F,0,N)
    v=vector(F,N)

    for r in range(rho):
        U=Matrix(F,N,0)
        for i in range(M):
            for j in range(alfa):
                U=U.augment(D.random_element().column())


        dim=ceil((r+1)*c/K)

        for i in range(dim):
            pot=(r+1)*c-(i+1)*K+K+b-1
            for j in range(N):
                v[j]=alpha[j]^pot*mult[j]
            U[:,(f-1)*alfa+i]+=v.column()


        for i in range(N):
            queries[i][r,:]=U[i,:]
    return queries
```

### A.5.2. Recuperación del archivo a partir de las respuestas

```
def PIRBetterBizantineDecodeFromResponses(C,b,z,r,alfa,R,Erasures):

    """
    This function recovers the desired file from the responses of the
        databse to the queries create by
        PIRBetterBizantineQueryMatrices() program
```

```
The inputs required for this function are the following:
    C: Linear code used to encode the distributed database to be
        queried
    b: Number of colluding servers that we want to protect against
    z: Number of byzantine servers that we want to protect against
    r: Number of non-responsive servers that we want to
        protect against
    alpha: Number of stripes in which each file is divided
    R: Matrix of responses from the servers of the database. Each
        column is the response from one server
    Erasures: Matrix of the same size as 'R' containing elements
        of the two element field. Positions with a one means that
        an erasure has been received in that position

The output of this program will be the file that we wanted in its
    original form
"""

N=C.length()
K=C.dimension()
c=N-K-b-2*z-r+1
if c<1:
    raise ValueError('No se puede realizar PIR para esos parámetos')




mcm=lcm(c,K)
a=mcm/K
rho=mcm/c

if a!=alfa:
    print("Number of stripes does not match the required number")
    return -1;

F=C.base_field()
FF=R[0][0,0].parent()

alpha=vector(FF,C.evaluation_points())
mult=C.column_multipliers()
prod=mult.pairwise_product(mult)

dim=c+K+b-1
Star=codes.GeneralizedReedSolomonCode(alpha,dim,prod)
decoder=codes.decoders.GRSErrorErasureDecoder(Star)

v=Matrix(FF,1,N)
File=Matrix(FF,alfa,K)
```

```
Response=Matrix(FF,rho,0)
 for i in range(N):
     Response=Response.augment(R[i])

 for r in range(rho):
     pot=(r+1)*c+K+b-2

     #Delete known values
     for i in range(c*r):
         banda=i//K
         column=i%K
         for j in range(N):
             v[0,j]=alpha[j]^(pot-i)*prod[j]
         Response[r,:]-=v*File[banda,K-1-column]



     #Decode the rest
     word=decoder.decode_to_message([Response.row(r), Erasures.row(r)])

     pos=c*r
     for i in range(c):

         banda=(pos+i)//K
         column=(pos+i)%K

         File[banda,K-1-column]=word[c+K+b-2-i]


 #Decode file
 file=DecodeToFile(File,F)
 return file
```

# Bibliografía

[1] Ruud Pelikan, Xin-Wen Wu, Stanislav Bulygin, Relinde Jurrius. *Codes, Criptology and Curves with Computer Algebra*, Cambridge University Press, 2017

[2] Carlos Munuera Gomez, Juan Gabriel Tena Ayuso. *Codificación de la Información*, Universidad de Valladolid, 1997

[3] Jørn Justesen, Tom Hohøldt. *A course in Error-correcting Codes.* European Mathematical Society, 2004

[4] W. Cary Huffman, Vera Pless, *Fundamentals of Error Correcting Codes*, Cambridge University Press, 2010

[5] R. Tajeddine, O.W. Gnilke, S. El Roayheb, *Private Information Retrieval From MDS Coded Data in Distributed Storage Systems*, IEEE Transactions on Information Theory 64 (2018) 7081-7093.

[6] Thomas M. Cover, Joy A. Thomas, *Elements of Information Theory*, John Willey & Sons Inc, 1991.

[7] R. Freij-Hollanti, O. Gnilke, C. Hollanti, D. Karpuk, *Private Information Retrieval from Coded Databases with Colluding Servers*, SIAM J. Appl. Algebra Geometry 1 ( 2017), 647–664

[8] J. Schur *Bemerkungen zur Theorie der beschränkten Bilinearformen mit unendlich vielen Veränderlichen*, Journal für die reine und angewandte Mathematik. 1911 (140): 1–28. 1911

[9] van Lint, J.H. ; Wilson, R.M., *On the minimum distance of cyclic codes*, IEEE Transactions on Information Theory. 1986 ; Vol. 32, No. 1. pp. 23-40

[10] R. Tajeddine, O.W. Gnilke, D. Karpuk, R. Freij-Hollanti, C. Hollanti, *Robust Private Information Retrieval from Coded Systems with Byzantine and Colluding Servers*, arXiv 1802.03731

[11] R. Tajeddine, O.W. Gnilke, D. Karpuk, R. Freij-Hollanti, , *Private Information Retrieval from Coded Storage Systems with Colluding, Byzantine, and Unresponsive Servers*, IEEE Transactions on Information Theory 65 (2019) 3898 - 3906

[12] SageMath, the Sage Mathematics Software System (Version 9.1), The Sage Developers, 2020, https://www.sagemath.org.