VNiVERSiDAD D SALAMANCA

Facultad de Ciencias
Grado en Matemáticas

Trabajo de Fin de Grado

Recuperación de información privada

Autora:
Noelia Seco Redondo

Tutor:
José Ignacio Iglesias Curto

Julio de 2024

VNiVERSiDAD D SALAMANCA

Facultad de Ciencias
Grado en Matemáticas

Trabajo de Fin de Grado

Recuperación de información privada

Autora:
Noelia Seco Redondo

Tutor:
José Ignacio Iglesias Curto

SECO REDONDO NOELIA - 71707078R

Firmado digitalmente por SECO REDONDO NOELIA - 71707078R
Fecha: 2024.07.04 21:38:53 +02'00'

Julio de 2024

VNiVERSiDAD D SALAMANCA
CAMPUS DE EXCELENCIA INTERNACIONAL

Facvltad D Ciencias
VNiVERSiDAD D SALAMANCA

Certificado del/los tutor/es TFG

D. José Ignacio Iglesias Curto, profesor del Departamento de Matemáticas de la Universidad de Salamanca,

HACE CONSTAR:

Que el trabajo titulado "Recuperación de información privada", que se presenta, ha sido realizado por Dña. Noelia Seco Redondo, con DNI 71707078R y constituye la memoria del trabajo realizado para la superación de la asignatura Trabajo de Fin de Grado en Matemáticas en esta Universidad.

Salamanca, a 4 de Julio de 2024

Fdo.: Firmado digitalmente por IGLESIAS CURTO JOSE IGNACIO - 70866916Y
Fecha: 2024.07.04 21:33:22 +02'00'

Índice general

Introducción

La criptografía es el estudio de las técnicas que permiten que dos partes se comuniquen en un ámbito de desconfianza, de manera que el receptor legítimo sea el único que tenga acceso a esa información. En oposición, se encuentra el criptoanálisis que estudia los métodos para que un tercero acceda a la información a la que no tiene autorización. La unión de ambos estudios es la criptología [20].

En este trabajo se estudiará una técnica concreta de la criptografía, la recuperación de información privada. Su denomincaión proviene de la traducción del inglés *Private Information Retrieval* de donde se obtienen sus siglas, PIR. Este procedimiento permite a un usuario obtener un registro concreto de una base de datos sin revelar el índice del registro requerido. Formalmente denotemos x a la base de datos y supongamos que es una cadena de n bits, $x = x_0 \dots x_{n-1}$. El usuario quiere obtener x_i sin filtrar ninguna información sobre el índice i [7] .

Esta técnica se utiliza en caso de que el usuario quiera mantener en secreto la información extraída. En caso contrario, simplemente le bastará con pedir directamente el registro deseado a la base de datos. Entonces, los protocolos PIR se usan para descargar registros sin filtrar ninguna información de los mismos a la base de datos.

Un protocolo PIR es seguro si el usuario obtiene el bit deseado, x_i, correctamente y sin filtrar ningún tipo de información acerca del índice $i \in \{0, \dots, n-1\}$. Esta técnica es útil, por ejemplo, en el caso de que un inversor quiera obtener cierta información sobre algún producto financiero, ubicado de una base de datos, sin que la entidad financiera, propietaria de la base de datos, detecte que información se está consultando [7].

El concepto de PIR fue introducido en 1995 por B. Chor, O. Goldreich y M. Sudan en [7] bajo una seguridad incondicional, es decir, el protocolo es seguro frente a terceros con capacidades computacionales y tiempo ilimitados. Para obtener esta seguridad, si la información está sólo en una base de datos, se necesitaría descargar toda la base. En este caso se comunicarían todos los n bits, lo que supone una complejidad de comunicación inaceptable. Ésta se puede reducir mediante la copia de la base de datos en k servidores con $k \geq 2$. En este trabajo se expondrán esquemas con complejidades de comunicación sublineales en n y alcanzando la seguridad deseada [17].

Fue más tarde, en 1997, cuando se consideró una modificación del PIR con seguridad incondicional al PIR computacional, abreviado como cPIR. En estos protocolos, las terceras partes, que son los servidores, sólo pueden amenazar la privacidad mediante operaciones acotadas computacionalmente por un límite superior. En estos casos la privacidad reside en problemas intratables con sus capacidades computacionales. Algunos de estos problemas son el cálculo de residuos cuadráticos módulo un número compuesto o las funciones hash. Estos esquemas consiguen una complejidad de comunicacion mejor que el PIR bajo seguridad incondicional. Incluso no necesitan la replicación de la base de

datos en varios servidores [17].

Siguiendo las premisas anteriores, el documento está organizado en tres capítulos.

- En el capítulo 1 constan los preliminares. En ellos están las definiciones básicas que se necesitarán para comprender el resto del documento.
- En el capítulo 2 se expondrán tres protocolos PIR con seguridad incondicional. En todos ellos se analizará si son seguros y su complejidad de comunicación total. Cabe destacar que cada uno de ellos va reduciendo la comunicación total en comparación con el anterior.
- En el capítulo 3 se expondrá el protocolo PIR con seguridad incondicional que obtiene la mejor comunicación total estudiada. Esto se alcanzará representando la base de datos con un polinomio y descomponiéndolo correctamente.
- En el capítulo 4 se expondrá un protocolo PIR con seguridad computacional. De este protocolo destacaremos que no será necesario replicar la base de datos en varios servidores, con sólo un servidor se alcanzará la seguridad.

Además, vamos a ilustrar los distintos protocolos con ejemplos. Todos ellos han sido realizados por mí, para ayudar a una mejor comprensión de cada protocolo PIR. Cabe resaltar, que están realizados sin sustituir los bits de la base de datos para visualizar claramente cómo se recupera el bit correcto.

Por último, hay cinco apéndices con ejemplos para ayudar a una mejor comprensión del documento. Se indica a lo largo del mismo cómo acudir a ellos.

Capítulo 1

Preliminares

Veamos unos conceptos e ideas necesarias para el desarrollo posterior del trabajo [7].

Comencemos describiendo con un poco más de detalle qué es un protocolo PIR. Lo primero que debemos ilustrar es la formalización de la base de datos.

Una base de datos, x, es un conjunto organizado de datos. En este trabajo por simplicidad supondremos que es una cadena binaria de longitud n. Por lo que, a partir de ahora la denotaremos como $x = x_0 \dots x_{n-1}$, dónde $x_i \in \mathbb{F}_2$, $\forall i \in \{0, \dots, n-1\}$. En múltiples protocolos PIR la base de datos estará replicada en varios servidores. Es decir, cada servidor tendrá una copia de la base de datos y podrá recibir consultas del usuario.

Los protocolos PIR permitirán a un usuario recuperar un bit de una base de datos, supongamos que es x_i, manteniendo en secreto el índice i frente a los servidores. A lo largo del documento, denotaremos U al usuario y $S_0, S_1, \dots, S_{k-1}$ a los k servidores.

Para recuperar el bit deseado, el usuario mandará una o varias consultas a cada servidor o sólo a algunos de ellos. Si se mandan consultas sólo a k' servidores con $k' < k$, se dice que hay un total de k servidores, pero que solo k' son participantes. Las consultas, en general, se formalizarán como aplicaciones, dónde en el espacio inicial se usará un vector aleatorio.

Posteriormente, cada servidor que ha recibido una o varias consultas devolverá una respuesta al usuario. Esta respuesta también se modelizará como una aplicación en la que en el espacio inicial constará la base de datos y la consulta recibida por el usuario.

Finalmente, el usuario recuperará el bit usando las respuestas de los servidores y los vectores aleatorios que utilizó para formar las consultas. Las definiciones de estas aplicaciones se verán explícitamente al inicio de cada sección.

Una vez visto el funcionamiento de un protocolo PIR, veamos las dos condiciones que debe cumplir para que sea seguro utilizarlo:

1. **Condición de privacidad.** El servidor no puede diferenciar consultas de bits de índices diferentes. Es decir, la probabilidad de recibir una consulta sobre un índice $i \in \{0, \dots, n-1\}$ tiene que ser la misma que para un índice $j \neq i$, con $j \in \{0, \dots, n-1\}$.

2. **Condición de corrección.** El usuario debe poder recuperar exactamete el bit x_i en el que está interesado utilizando las respuestas de los servidores y los vectores aleatorios. Es decir, si el bit recuperado es x_j con $j \neq i$ sería incorrecto y por tanto no sería un protocolo PIR.

Estas propiedades también las desarrollaremos en profundidad en cada sección.

1.1. Terminología básica

A lo largo del trabajo se denotará con $\mathbb{Z}_n$ al conjunto de los restos al dividir entre $n \in \mathbb{N}$. Si n es un número primo, $\mathbb{Z}_n$ es un cuerpo y lo denotaremos como $\mathbb{F}_n$.

Veamos algunas definiciones que se usarán a lo largo del documento [4, 7, 10].

Definición 1.1. *La complejidad total de comunicación de un esquema PIR es la suma del máximo número de bits enviados entre el usuario y los servidores. En general, dependerá de n y k.*

Definición 1.2. *Se dice que un protocolo PIR confabula si los servidores se alían para quebrar la privacidad del usuario.*

En todos los protocolos incluidos en este trabajo los servidores no confabularán.

La siguiente definición es interesante sobre todo cuando apliquemos la recursión a los protocolos para obtener complejidades totales de comunicación menores.

Definición 1.3. *Un esquema PIR será de una ronda si para que el usuario obtenga el bit deseado manda una o varias consultas a cada servidor y posteriormente recibe una respuesta de cada uno obteniendo el bit de interés. Si después de la respuesta de los servidores, el usuario mandase nuevas consultas, de las que obtendría respuestas, sería de varias rondas.*

Observación 1.4. Durante el trabajo, se utilizarán numerosas veces conjuntos del tipo $\{1, \ldots, n\}$, para agilizar su escritura fijamos la notación $[n] := \{1, \ldots, n\}$.

El documento se va a dividir entre los protocolos PIR con seguridad incondicional o teórica y aquellos con seguridad computacional. Por ello, veamos sus definiciones:

Definición 1.5. *Un esquema PIR tiene seguridad incondicional o teórica si suponiendo que los servidores son computacionalmente ilimitados, después de la comunicación entre el usuario y los servidores, éstos últimos no obtienen ninguna información del índice del bit requerido. Para ello las consultas mandadas a los servidores no pueden revelar ningún tipo de información del índice del bit que quiere recuperar el usuario.*

Definición 1.6. *Un esquema PIR tiene seguridad computacional (cPIR) si suponiendo que los servidores son computacionalmente limitados, después de la comunicación entre el usuario y los servidores, éstos últimos no obtienen ninguna información del índice del bit requerido. Para ello hay que asegurar que cualquier cómputo para obtener cualquier información sobre el índice deseado es más costoso que su limitación computacional.*

Las dos definiciones anteriores junto con la Definición 1.1 son claves para el desarrollo del trabajo. El objetivo principal de un protocolo PIR es conservar las propiedades de privacidad y corrección. Una vez que se han alcanzado, el segundo objetivo es reducir la comunicación total al mínimo posible.

En el caso de PIR con seguridad incondicional, si el número de servidores es $k = 1$, la única solución es la trivial. La solución trivial consta de mandar la base de datos entera,

que tiene n bits. Su comunicación total es $n = \mathcal{O}(n)$ bits. Para reducir este orden de complejidad de comunición en el caso de que los servidores sean computacionalmente ilimitados la solución será replicar la base de datos en k servidores, con $k \geq 2$.

En el caso de que sean computacionalmente limitados se conseguirán protocolos con complejidades menores incluso para $k = 1$ servidor, porque las consultas a los servidores sobre cualquier índice sólo tienen que ser indistinguibles bajo la limitación computacional de los servidores. A lo largo del documento se estudiarán protocolos de ambos tipos analizando su complejidad, así como las propiedades de privacidad y corrección.

1.2. Códigos

Vamos a introducir ligeramente la Teoría de Códigos ya que se usará en algún protocolo.

La transmisión de información se realiza entre un emisor y un receptor a través de un canal. La Teoría de Códigos estudia métodos para recuperar dicha información después de su paso por el canal ruidoso. Para ello se codifica la información añadiéndole una redundancia. Después del paso por el canal, se analiza si ha habido errores en la transmisión. En caso de que no, se elimina la redundancia para obtener el mensaje. En caso de que sí, algunos se pueden corregir y obtener el mensaje. Principalmente se estudian el proceso de codificación y el de detección y corrección de errores que es la decodificación. Veamos algunos conceptos más técnicos para fijar su notación.

Considerando el alfabeto, $\mathbb{F}_q$ y la aplicación de codificación $\psi_C : \mathbb{F}_q^k \hookrightarrow \mathbb{F}_q^n$, un código lineal, C, es un $\mathbb{F}_q$ - subespacio vectorial de $\mathbb{F}_q^n$. La matriz asociada a ψ_C es la matriz generadora del código C. Para detectar los errores, se utiliza la matriz de control del código C que se denota H. Es la correspondiente al subespacio ortogonal del subespacio del código respecto del producto escalar habitual. Asimismo, también usaremos códigos no lineales, que no serán $\mathbb{F}_q$ - subespacios vectoriales. Las siguientes definiciones serán útiles también.

Definición 1.7. *Sean $s = (s_1, \ldots, s_n)$ y $t = (t_1, \ldots, t_n)$ dos vectores. Su distancia de Hamming es el número de componentes distintas que tienen. Se denota $d_H(s,t) := |\{i/s_i \neq t_i\}| = |\{i/s_i - t_i \neq 0\}|$.*

Definición 1.8. *Sea $s = (s_1, \ldots, s_n)$ un vector. Su peso de Hamming es el número de componentes distintas de 0 que tiene. Se denota $w_H(s) := |\{i/s_i \neq 0\}|$.*

Capítulo 2

PIR seguridad incondicional

En este capítulo se verán diversos protocolos PIR que tienen seguridad incondicional. Como el objetivo es reducir al máximo la comunicación total entre usuario y servidor, es necesario que el número de servidores, k, que tienen una copia de la base de datos sea mayor o igual a 2. Recordemos que su notación es $S_0, \ldots, S_{k-1}$.

Supongamos que el usuario, U, quiere obtener x_i sin que los servidores obtengan ninguna información sobre el índice $i \in \mathbb{Z}_n$. Entonces definimos un modelo PIR de la siguiente manera [7, 4].

Definición 2.1. *Denotemos $x = x_0 \ldots x_{n-1}$ la base de datos de longitud n donde $x_j \in \mathbb{F}_2$, $\forall j \in \mathbb{Z}_n$. Un modelo PIR de k servidores está formado por las siguientes aplicaciones:*

- $Q_0, \ldots, Q_{k-1} : \mathbb{Z}_n \times \mathbb{Z}_m^r \longrightarrow \mathbb{Z}_m^q$. *Estas aplicaciones se denominan aplicaciones de consulta.*
- $A_0, \ldots, A_{k-1} : \mathbb{Z}_2^n \times \mathbb{Z}_m^q \longrightarrow \mathbb{Z}_2^a$. *Estas aplicaciones se denominan aplicaciones de respuesta.*
- $R : \mathbb{Z}_n \times \mathbb{Z}_m^r \times (\mathbb{Z}_2^a)^k \longrightarrow \mathbb{Z}_2$. *Esta aplicación modela la recuperación del bit deseado.*

Veamos que significan cada una de las aplicaciones que forman el protocolo PIR.

- Cada Q_j modela la consulta o consultas que hace U al servidor S_j con $j \in \mathbb{Z}_k$. En cada consulta se toma el índice $i \in \mathbb{Z}_n$ correspondiente al bit x_i que desea recuperar y un vector $u \in \mathbb{Z}_m^r$ que es un vector aleatorio de r componentes de $\mathbb{Z}_m$, con $r \in \mathbb{N}$. En cada protocolo PIR se especificará el valor de m. Las consultas que recibe cada servidor son vectores de $\mathbb{Z}_m^q$, no tienen por qué tener los mismo número de componentes que el vector aleatorio. Por ello se utiliza la letra $q \neq r$, con $q \in \mathbb{N}$.
- Cada A_j modela la respuesta que devuelve el servidor, S_j, con $j \in \mathbb{Z}_k$ a U. Para crear esta respuesta cada servidor utiliza la base de datos y la consulta enviada por el usuario U. Aunque el servidor reciba varias consultas de U para la respuesta que devuelve solo usará una de ellas[1]. El conjunto inicial es el producto cartesiano de $\mathbb{Z}_2^n$ y $\mathbb{Z}_m^q$, porque la base de datos, x, pertenece a $\mathbb{Z}_2^n$ y las consultas al conjunto $\mathbb{Z}_m^q$.

[1]Esta idea se entenderá mejor cuando veamos el protocolo PIR de Ambainis, en el que para mejorar la complejidad total se realizará varias veces el protocolo de manera recursiva. Para ello se enviarán varias consultas a cada servidor pero para la respuesta final sólo usará una.

La respuesta que manda cada servidor, S_j, pertenece a $\mathbb{Z}_2^a$ porque devolverá uno o varios bits de la base de datos, según el tipo de protocolo que se esté utilizando.

- La aplicación R modela la recuperación del bit, x_i, querido por usuario, dónde $\mathbb{Z}_m^r$ es un vector aleatorio y cada uno de los conjuntos $(\mathbb{Z}_2^a)$ es la respuesta que ha devuelto el servidor S_j con $j \in \mathbb{Z}_k$. El conjunto final es $\mathbb{Z}_2$ ya que su imagen será el bit, 0 o 1, que le interesa a U.

Las aplicaciones explicadas forman el protocolo PIR, pero tienen que cumplir obligatoriamente las siguientes dos peopiedades:

1. Un modelo PIR se dice que verifica la propiedad de **corrección** si para cada $i \in \mathbb{Z}_n$, $u \in \mathbb{Z}_m^r$ se verifica:

$$P(R(i, u, A_0(x, Q_1(i, u)), \ldots, A_{k-1}(x, Q_k(i, u))) = x_i) = 1$$

   Es decir, la probailidad de que U recupere el bit x_i que desea es 1.

2. Un modelo PIR se dice que verifica la propiedad de **privacidad** si para cualesquiera índices $h, m \in \mathbb{Z}_n$, $j \in \mathbb{Z}_k$ y $u \in \mathbb{Z}_m^r$ escogido uniformemente.

$$P(Q_j(h, u)) = P(Q_j(m, u))$$

   Es decir, la probabilidad de que el servidor S_j reciba una consulta es la misma sea cual sea el índice de interés de U.

Definición 2.2. *Se dice que un protocolo es de una ronda si inicialmente U manda una o varias consultas a cada servidor S_j, con $j \in \mathbb{Z}_k$. Cada uno de ellos responde aplicando la función A_j y finalmente U recupera el bit deseado mediante la aplicación R. En caso contrario, será de varias rondas si después de las respuestas recibidas por los servidores, U envía consultas nuevas, que son contestadas de nuevo por los servidores. El protocolo tendrá tantas rondas como veces mande U consultas a los servidores en distinto tiempo.*

Una vez definido con detalle qué es un protocolo PIR podemos definir su complejidad de comunicación total.

Definición 2.3. *La complejidad de comunicación total en un protocolo PIR, P, de k servidores con una base de datos de n bits es una función de k y n que mide el máximo número de bits comunicados entre el usuario y los k servidores. Si denotamos $Q_P(n, j)$ el máximo número de bits mandados del usuario a cada servidor S_j con $j \in \mathbb{Z}_k$ y $A_P(n, j)$ el máximo número de bits mandados de cada servidor S_j con $j \in \mathbb{Z}_k$ al usuario, la complejidad del protocolo, $C_P(n, k)$, es $C_P(n, k) = \sum_{j=1}^{k} Q_P(n, j) + \sum_{j=1}^{k} A_P(n, j)$.*

Normalmente, calcularemos la suma de todos los bits intercambiados entre usuario y servidores y posteriormente el orden de complejidad en función de k y n.

Observación 2.4. Es importante resaltar que se desea obtener protocolos cuya comunicación total sea sublineal en n. En general, las complejidades de comunicación serán de la forma $\mathcal{O}(c_1 n^{c_2})$, es decir, tendremos dos factores c_1 y n^{c_2}. Será clave reducir ambos, pero es más deseable reducir el segundo disminuyendo c_2 lo máximo posible, esto es porque n, que es el número de bits de la base de datos, es un número muy grande.

Observación 2.5. El objetivo de este capítulo exponer diferentes protocolos PIR cuyos servidores estén ilimitados computacionalmente, comprobar que verifiquen las condiciones de corrección y privacidad y analizar las complejidades de comunicación. Si solo hay un servidor, la única solución para que U recupere el bit es la trivial, es decir, mandar los n bits que forman la base de datos. Por este motivo, estudiaremos tres protocolos en este capítulo en los que mediante la replicación de la base de datos en varios servidores, alcanzaremos complejidades sublineales en n. Se estudiará otro protocolo de manera especial en el siguiente capítulo.

Para los primeros tres protocolos necesitamos la siguiente operación.

Definición 2.6. *Se define la operación o-exclusivo o XOR de dos bits x y z:*

$$x \oplus z := \begin{cases} 1 & si \ \ x \neq z \\ 0 & si \ \ x = z \end{cases}$$

Definición 2.7. *Se define la operación o-exclusivo o XOR de un conjunto W y un elemento a como:*

$$W \oplus \{a\} := \begin{cases} W \cup \{a\} & si \ a \notin W \\ W \backslash \{a\} & si \ a \in W \end{cases}$$

2.1. PIR de Chor et al. para k servidores

Este protocolo fue estudiado por B. Chor, E. Kushilevitz, O. Goldreich, y M. Sudan en 1995 [7, 6]. Permite a U recuperar el bit de interés mediante el envío de consultas a $k = 2^d$ servidores con $d \geq 1$. La elección de $k = 2^d$ viene dada porque asociaremos los k servidores con las tuplas de $\mathbb{F}_2^d$. Esta asociación se va a realizar escribiendo el subíndice de cada servidor $S_0, \ldots, S_{k-1}$, que es $\alpha \in \mathbb{Z}_k$, en base 2. Entonces, cada subíndice $\alpha \in \mathbb{Z}_k$ en base 10, lo asociaremos con $(\alpha_1 \ldots \alpha_d)_2$ que está en base 2. Más adelante se verá la importancia de que los subíndices de los servidores y los de los bits tengan el mismo número de componentes. Como no hay el mismo número de servidores y de bits y los subíndices de los servidores ya se han escrito en base 2 mediante d-tuplas, entonces se define $l = \lceil \sqrt[d]{n} \rceil$ que será la base en que se escribirán los índices de cada elemento de la base de datos y por lo tanto $n \in \mathcal{O}(l^d)$. Así, se escribe cada subíndice $j \in \mathbb{Z}_n$, correspondiente al bit $x_j \in \mathbb{F}_2$ que está en base 10, en base l, obteniéndose $(j_1 \ldots j_d)_l$, con $j_1, \ldots, j_d \in \mathbb{Z}_l$.

Observación 2.8. Ahora nos podemos referir a cada servidor concreto, S_α, mediante su subíndice $\alpha \in \mathbb{Z}_k$ que está en base 10 ó $(\alpha_1 ... \alpha_d)_2$ que está en base 2. En este documento, mientras no haya lugar a confusión, para la notación en base 2 utilizaremos indistintamente tanto $(\alpha_1 ... \alpha_d)_2$ como la d-tupla $(\alpha_1, ..., \alpha_d) \in \mathbb{F}_2^d$. La razón es que numerosas veces se utilizarán los dígitos en base 2 por separado no todos ellos como un número. Lo mismo sucede con los índices $j \in \mathbb{Z}_n$, correspondientes a los bits $x_j \in \mathbb{F}_2$, que se escribirán en base l indistintamente como $(j_1 \ldots j_d)_l$ o como la d- tupla $(j_1, \ldots, j_d) \in \mathbb{Z}_l^d$. Por lo que a cada servidor con subíndice $\alpha \in \mathbb{Z}_k$ lo podremos denotar como S_α, $S_{(\alpha_1,\ldots,\alpha_d)}$ ó $S_{\alpha_1 \ldots \alpha_d}$ y cada bit con subíndice $j \in \mathbb{Z}_n$ como x_j, $x_{(j_1,\ldots,j_d)}$ ó $x_{j_1 \ldots j_d}$. En la anterior notación y a lo largo del trabajo se ha obviado el subíndice 2 indicando base 2 en los servidores y el l indicando base l en los índices de los bits. Aún así, para diferenciarlos perfectamente nos

referiremos a los índices de los servidores con letras griegas y a los índices de los bits con letras latinas. Esta notación explicada en esta observación se utilizará a lo largo de todo el capítulo.

Veamos el procedimiento del esquema:

1. U quiere obtener el bit x_i sin revelar a los servidores el índice $i \in \mathbb{Z}_n$. En concreto, asociamos el índice i con la d-tupla $(i_1, \ldots, i_d) \in \mathbb{Z}_l^d$. Inicialmente U escoge d subconjuntos aleatorios e independientes: $W_1^0, W_2^0, \ldots, W_d^0 \subseteq \mathbb{Z}_l$. A partir de estos conjuntos, U define $W_j^1 = W_j^0 \oplus \{i_j\}$, $\forall j \in [d]$. Una vez construidos los $2d$ subconjuntos, a cada servidor, $S_{(\alpha_1,\ldots,\alpha_d)}$, con $(\alpha_1, ..., \alpha_d) \in \mathbb{F}_2^d$, se le mandarán los d subconjuntos $W_1^{\alpha_1}, W_2^{\alpha_2}, \ldots, W_d^{\alpha_d}$ correspondientes a su d-tupla.

2. Cada servidor, $S_{(\alpha_1,\ldots,\alpha_d)}$, recibe sus d subconjuntos y devuelve el resultado de la siguiente operación XOR.

$$\bigoplus_{j_1 \in W_1^{\alpha_1}, \ldots, j_d \in W_d^{\alpha_d}} x_{j_1,\ldots,j_d}$$

   Este cálculo lo realiza cada servidor y manda el bit resultante al usuario.

3. U hace la operación XOR con todos los bits recibidos. El resultado es la recuperación de x_i.

Observación 2.9. Analicemos la relación del procedimiento del protocolo con las aplicaciones formales de un protocolo PIR descritas en la Definición 2.1.

Las aplicaciones de consulta Q_j con $j \in \mathbb{Z}_k$, en este caso serían $Q_{(\alpha_1,\ldots,\alpha_d)}$ con $(\alpha_1, ..., \alpha_d) \in \mathbb{F}_2^d$, porque los k servidores se asocian con las tuplas de $\mathbb{F}_2^d$.

$Q_{(\alpha_1,\ldots,\alpha_d)} : \mathbb{Z}_n \times \mathbb{Z}_l^{ld} \longrightarrow \mathbb{Z}_l^{ld}$, dónde en $\mathbb{Z}_n$ el usuario toma i porque es el índice del bit que desea, en $\mathbb{Z}_l^{ld}$ el vector aleatorio que está formado por los conjuntos aleatorios $W_j^0 \subseteq \mathbb{Z}_l$ con $j \in [d]$, como hay d conjuntos y cada uno tiene hasta l elementos de $\mathbb{Z}_l$ el vector aleatorio tiene longitud ld. La imagen es la consulta que se realiza al servidor $S_{(\alpha_1,\ldots,\alpha_d)}$ que son los $W_1^{\alpha_1}, W_2^{\alpha_2}, \ldots, W_d^{\alpha_d}$ pertenecientes a $\mathbb{Z}_l^{ld}$.

Las aplicaciones de respuesta A_j con $j \in Z_k$, que en este caso serían $A_{(\alpha_1,\ldots,\alpha_d)}$ con $(\alpha_1, ..., \alpha_d) \in \mathbb{F}_2^d$.

$A_{(\alpha_1,\ldots,\alpha_d)} : \mathbb{Z}_2^n \times \mathbb{Z}_l^{ld} \longrightarrow \mathbb{Z}_2$, dónde $\mathbb{Z}_2^n$ representa la base de datos, del conjunto $\mathbb{Z}_l^{ld}$ se toma la consulta que ha recibido el servidor $S_{(\alpha_1,\ldots,\alpha_d)}$ de la aplicación $Q_{(\alpha_1,\ldots,\alpha_d)}$. El conjunto final es $\mathbb{Z}_2$ con $a = 1$ porque cada servidor devuelve 1 bit definido mediante la operación XOR del paso 2.

Finalmente, la aplicación R queda reflejada en el paso 3.

$R : \mathbb{Z}_n \times \mathbb{Z}_l^{ld} \times (\mathbb{Z}_2)^k \longrightarrow \mathbb{Z}_2$, dónde en $\mathbb{Z}_n$ se toma el índice i del bit que se desea, en $\mathbb{Z}_l^{ld}$ el vector aleatorio definido en la aplicación de consulta y en $(\mathbb{Z}_2)^k$ los k bits obtenidos en las aplicaciones de respueta de los k servidores. U realiza la operación XOR de todos los bits de las respuestas y obtiene el bit deseado.

Analicemos ahora la corrección, privacidad y complejidad de este protocolo.

Proposición 2.10. *En este protocolo se cumple la propiedad de corrección.*

Demostración. Para probar la corrección hay que demostrar dos propiedades respecto al conjunto de bits que recibe U de todos los servidores. En la demostración se hará abuso de lenguaje diciendo que los servidores reciben índices de bits de la base de datos cuando en realidad reciben subconjuntos a partir de los cuales mediante la operación XOR definida en el paso 2 se obtienen los bits de la base de datos.

La primera, que el bit $x_{(i_1,\dots,i_d)}$, con $(i_1,\dots,i_d) \in \mathbb{Z}_l^d$ está un número impar de veces entre los bits recibidos. La segunda, que el resto de índices se encuentran en un número par de veces. Consecuentemente, al realizar la operación XOR quedará solo el bit $x_{(i_1,\dots,i_d)}$, como se desea. Esto se deduce de la Definición 2.6 de la operación XOR para bits, si hay dos bits cuyos subíndices son iguales al realizar XOR entre ellos se cancelan.

El subíndice $(i_1,\dots,i_d)$ no sólo está un número impar de veces sino que está sólo una vez. Esto se deduce de que para cada $j \in [d]$ el índice i_j sólo está en un subconjunto de la pareja (W_j^0, W_j^1). Entonces, i_1 pertenecerá a $W_1^{\beta_1}$ con $\beta_1 \in \mathbb{F}_2$, i_2 pertenecerá a $W_2^{\beta_2}$ con $\beta_2 \in \mathbb{F}_2$ así hasta i_d que pertenecerá a $W_d^{\beta_d}$ con $\beta_d \in \mathbb{F}_2$. Es decir, sólo el servidor cuyo subíndice es $(\beta_1,...,\beta_d)$ va a recibir el bit cuyo subíndice es $(i_1,\dots,i_d)$ completo. En la respuesta de dicho servidor estará $x_{(i_1,\dots,i_d)}$ y en ninguna respuesta de ningún servidor más.

Veamos el razonamiento para el resto de índices. La diferencia entre los subconjuntos mandados a cualquier servidor es la presencia o no de algún componente de $(i_1,\dots,i_d) \in \mathbb{Z}_l^d$. Entonces, comprobemos que cualquier bit con índice $(j_1,\dots,j_d) \neq (i_1,\dots,i_d)$ se encontrará un número par de veces como sumando de los símbolos que recibe U. Supongamos que $j_t \neq i_t$, entonces si el índice $(j_1,\dots,j_d)$ lo recibe el servidor cuyo subíndice es $(\gamma_1,...,\gamma_{t-1},0,\gamma_{t+1},\dots,\gamma_d)$ también lo recibirá el servidor con subíndice $(\gamma_1,...,\gamma_{t-1},1,\gamma_{t+1},\dots,\gamma_d)$. En efecto, si el componente $j_t \neq i_t$, entonces si $j_t \in W_t^0$ también $j_t \in W_t^1$. Es decir, cada subíndice $(j_1,\dots,j_d)$ con al menos una componente distinta de $(i_1,\dots,i_d)$ lo va a recibir siempre un número par de servidores y éstos van a usar el bit correspondiente como sumando del bit que devuelven. Si todas las componentes de $(j_1,\dots,j_d)$ fuesen distintas de $(i_1,\dots,i_d)$, como el número de servidores que hay es par, ya que $k = 2^d$, cuando U haga XOR de todos los bits recibidos cada bit que no sea el deseado va a ser sumando de este XOR un número par de veces y los bits con subíndice igual se cancelan dos a dos. Por lo que concluimos que usando el protocolo anterior se obtiene x_i con probabilidad igual a 1. □

Proposición 2.11. *En este protocolo se cumple la propiedad de privacidad.*

Demostración. Cada servidor recibe d subconjuntos distribuidos uniforme e independientemente. Entonces, recibida una consulta, no puede deducir ninguna información sobre el índice requerido por U, es decir, la probabilidad de que una consulta sea sobre el índice de interés $j \in \mathbb{Z}_n$ o sobre $m \neq j \in \mathbb{Z}_n$ es la misma. □

Proposición 2.12. *La complejidad de comunicación de este protocolo es del orden* $\mathcal{O}(kn^{\frac{1}{\log_2 k}})$.

Demostración. U envía d subconjuntos de $\mathbb{Z}_l$ a cada uno de los k servidores. Por lo tanto cada subconjunto puede tener hasta l elementos de $\mathbb{Z}_l$. Es decir en total se envían kdl elementos. De cada servidor recibe 1 bit. Como $k = 2^d$ y $n \in \mathcal{O}(l^d)$, la comunicación total sería $k(d\sqrt[d]{n}+1) \in \mathcal{O}(k\sqrt[d]{n}) = \mathcal{O}(kn^{\frac{1}{\log_2 k}})$, porque $d < k$. □

Observación 2.13. En el momento de poner el protocolo en práctica se conoce el número total de bits de la base de datos, n, faltaría fijar d. Como se desea que la comunicación total sea la menor posible, la manera de fijar d sería obteniendo el valor k que minimiza el valor de la complejidad de comunicación, $kn^{\frac{1}{\log_2 k}}$. Una vez obtenido el valor k, se obtiene d porque $k = 2^d$ y también l porque $l = \lceil \sqrt[d]{n} \rceil$ y ya se podría desarrollar el protocolo. Así pues el valor óptimo de d varía según el número de bits de la base de datos. Si por razones logísticas el número de servidores está fijado a priori de comenzar el protocolo, utilizaríamos de los servidores disponibles la cantidad que más nos convendría minimizando la complejidad de comunicación.

2.2. PIR de Chor et al. basado en códigos de recubrimiento

Este modelo PIR fue estudiado por B. Chor, E. Kushilevitz, O. Goldreich, y M. Sudan en 1995 [7, 6]. En él vamos a mejorar la comunicación total del anterior protocolo reduciendo el número de servidores que participan, es decir, aquellos que reciben consultas y envían respuestas. Se realizará utilizando códigos de recubrimiento, de manera que en el protocolo sólo participarán k' servidores, con $k' < k$. Los k' servidores enviarán las respuestas que les corresponden como en el anterior protocolo y además, simularán a los $k - k'$ servidores restantes. En esta sección expondremos cómo se realizará esto. Veamos primero qué es un código de recubrimiento:

Definición 2.14. *Sean $r, n, m \in \mathbb{N}$ con $n \geq 2$. Un código $C \subseteq \mathbb{Z}_n^m$ es de recubrimiento con radio de recubrimiento r si para cualquier $u \in \mathbb{Z}_n^m$ existe una palabra del código $c \in C$ tal que $d_H(u, c) \leq r$. Dicho de otra manera, si el código C está formado por las palabras $\{c_1, \ldots, c_t\}$ entonces $\mathbb{Z}_n^m \subseteq \bigcup_{i=1}^{t} B(c_i, r)$.*

Observación 2.15. En la definición anterior n no tiene porque ser primo, por lo que el código no se ha definido sobre un cuerpo. Entonces, se utilizarán códigos de recubrimiento lineales y no lineales.

En este protocolo, el número total de servidores incluyendo los participantes y los simulados seguirá siendo $k = 2^d$ con $d \geq 1$, porque los seguiremos asociando con las tuplas de $\mathbb{F}_2^d$. Se continúa con el mismo criterio para la notación de los índices de los servidores y de los bits que en el protocolo anterior. La longitud de las tuplas de ambos tiene que ser la misma por lo que los subíndices de los bits se escriben en base $l = \lceil \sqrt[d]{n} \rceil$.

Este protocolo va a ser muy similar al anterior, la única diferencia es que de los k servidores totales sólo participarán en el protocolo k', con $k' < k$. Es decir, sólo k' recibirán consultas y devolverán respuestas, a partir de estos servidores se simularán al total. Por ello, el objetivo es encontrar una forma eficiente de conseguirlo, la herramienta que usaremos son los códigos de recubrimiento. Usaremos códigos de recubrimiento de k' palabras, que se corresponderán con los k' servidores participantes. Como escribimos los índices de los servidores en base 2, estos códigos serán binarios. Además, veremos en la Observación 2.20 que para obtener la menor complejidad nos centraremos en aquellos que tienen radio de recubrimiento igual a 1.

Observación 2.16. En Teoría de Códigos se denota con d a la distancia mínima de un código. En nuestro caso d seguirá siendo la dimensión de los vectores con los que

denotamos en base 2 a los índices de los servidores y en base l a los índices de los bits de la base de datos. En ningún momento se utilizará d como la distancia mínima del código.

Utilizando este protocolo, U quiere obtener el bit x_i correspondiente al índice $i \in \mathbb{Z}_n$ que en base l es $(i_1, \ldots, i_d) \in \mathbb{Z}_l^d$. Veamos su funcionamiento:

1. Como se ha mencionado antes sólo participarán en el protocolo PIR k' servidores, en vez de los k totales que hay. Cada servidor participante se va a asociar con una palabra del código de recubrimiento $C \subseteq \mathbb{Z}_2^d$. Por lo tanto, C estará formado por las palabras $\{\alpha_1, \ldots, \alpha_{k'}\}$, con $\alpha_j = (\alpha_{j_1}, \alpha_{j_2}, \ldots, \alpha_{j_d}) \in \mathbb{Z}_2^d$, dónde $j \in [k']$. Y consecuentemente los servidores participantes serán $S_{\alpha_1}, S_{\alpha_2}, \ldots, S_{\alpha_{k'}}$. De esta manera, U escoge aleatoria e independientemente los subconjuntos $W_1^0, W_2^0, \ldots, W_d^0 \subseteq \mathbb{Z}_l$ y define $W_j^1 = W_j^0 \oplus \{i_j\}$, $\forall j \in [d]$, manda a cada servidor S_{α_j}, con $j \in [k']$ los subconjuntos $W_1^{\alpha_{j_1}}, W_2^{\alpha_{j_2}}, \ldots, W_d^{\alpha_{j_d}}$. Este primer paso es el mismo que el del anterior protocolo pero reduciendo el número de servidores a los que U envía consultas, de k a k' servidores.

2. Cada servidor participante, S_{α_j}, con $j \in [k']$ realiza la operación XOR de $x_{j_1,\ldots,j_d}$, dónde $j_h \in W_h^{\alpha_{j_h}}$ para cada $h \in [d]$ y devuelve ese bit a U. Este paso también es igual que en el anterior protocolo, los servidores participantes devuelven un bit realizando la operación XOR con los índices que han recibido.

   Además, cada servidor participante, S_{α_j}, con $j \in [k']$ simula a los servidores que estén a distancia de Hamming 1 de él. Es decir, simula a todos los servidores $S_{(\beta_1,\ldots,\beta_d)}$ con $(\beta_1, ..., \beta_d) \in \mathbb{F}_2^d$ tales que $d_H((\beta_1, ..., \beta_d), (\alpha_{j_1}, ..., \alpha_{j_d})) = 1$. En total, cada servidor, S_{α_j}, simula a d servidores, porque hay d vectores de $\mathbb{F}_2^d$ a distancia 1 de $(\alpha_{j_1}, \alpha_{j_2}, \ldots, \alpha_{j_d}) \in \mathbb{F}_2^d$. Llamemos a los servidores que simula cada uno, $S_{\alpha_j^1}, \ldots, S_{\alpha_j^d}$, dónde el subíndice α_j^s es igual a $(\alpha_{j_1}, \alpha_{j_2}, \ldots, \alpha_{j_d}) \in \mathbb{F}_2^d$ cambiando el bit α_{j_s}, si es un 0 por un 1 y viceversa.

   Para simularlos deberá mandar el bit que hubiesen devuelto dichos servidores. El sevidor S_{α_j} conoce los subconjuntos $W_1^{\alpha_{j_1}}, W_2^{\alpha_{j_2}}, \ldots, W_d^{\alpha_{j_d}}$ de $\mathbb{Z}_l$ y necesitaría conocer el índice $(i_1, \ldots, i_d)$ del bit que desea U para construir los subconjuntos $\overline{W}_t^{\alpha_{j_t}} = W_t^{\alpha_{j_t}} \oplus i_t$, $\forall t \in [d]$. Así simularía a cada $S_{\alpha_j^s}$ con $s \in [d]$ realizando la operación XOR de $x_{j_1,\ldots,j_d}$, dónde $j_h \in W_h^{\alpha_{j_h}}$ para cada $h \in [d] \backslash s$ y $j_s \in \overline{W}_s^{\alpha_{j_s}}$. Esto no es posible ya que si los servidores S_{α_j} con $j \in [k']$ conocen $(i_1, \ldots, i_d) \in \mathbb{Z}_l^d$ se rompería la propiedad de privacidad.

   Por ello para simular a cada $S_{\alpha_j^s}$ con $s \in [d]$, S_{α_j} realiza la operación XOR de $x_{j_1,\ldots,j_d}$ un total de l veces, dónde $j_h \in W_1^{\alpha_{j_h}}$ para cada $h \in [d] \backslash s$ y $j_s \in W_s^{\alpha_{j_s}} \oplus \{r\}$ variando $r \in \mathbb{Z}_l$ en cada una de las l respuestas. Envía los l bits ordenados, es decir, primero el correspondiente a $W_s^{\alpha_{j_s}} \oplus \{0\}$, luego el $W_s^{\alpha_{j_s}} \oplus \{1\}$ y así hasta $W_s^{\alpha_{j_s}} \oplus \{l-1\}$. Es decir, en total cada servidor S_{α_j} manda l bits para simular a cada $S_{\alpha_j^s}$ con $s \in [d]$.

3. El usuario sólo realizará la operación XOR de los bits que correspondan a aquellos que hubiesen mandado los k servidores. Esto es posible ya que cada servidor S_{α_j} con $j \in [k']$ manda los bits en orden y U conoce el índice $(i_1, \ldots, i_d) \in \mathbb{F}_2^d$.

Observación 2.17. La relación del procedimiento del protocolo con las aplicaciones formales de un protocolo PIR descritas en la Definición 2.1, se haría de manera análoga a la realizada en la Observación 2.9 del protocolo anterior.

Proposición 2.18. *En este protocolo se cumplen las propiedades de privacidad y corrección.*

Demostración. La privacidad del protocolo se cumple ya que cada servidor recibe varios subconjuntos de $\mathbb{Z}_l$ distribuidos uniforme e independientemente de los demás.

La corrección del protocolo también se cumple. U sólo hará XOR de los bits correspondientes a los servidores participantes y de los que hubiesen enviado los servidores simulados, esto lo puede discriminar porque conoce el índice de cada uno de los simulados y el orden en que le han llegado los bits de cada vector simulado, tanto los que tiene que usar como los que no. Los bits de los que realiza la operación XOR corresponden con los que hubiesen mandado los k servidores. Es decir, hace XOR de los mismos bits que en el esquema anterior y por ello la corrección de este esquema se deduce de la corrección del anterior explicada en la Proposición 2.1. □

Teorema 2.19. *Sean $d, k' \in \mathbb{N}$ tal que existe un código de recubrimiento $C \subseteq \mathbb{Z}_2^d$ formado por las palabras $\{\alpha_1, \ldots, \alpha_{k'}\}$ con radio de recubrimiento 1. Entonces, existe un protocolo PIR para k servidores cuya comunicación total es $\mathcal{O}(k'dn^{\frac{1}{d}})$.*

Demostración. Veamos el número total de bits intercambiados entre el usuario y los servidores que participan en el protocolo. U manda d subconjuntos de $\mathbb{Z}_l$ a cada servidor participante, como hay k' servidores participantes, en total son $k'dl \in \mathcal{O}(k'dn^{\frac{1}{d}})$ bits. En la anterior igualdad se ha usado que $n \in \mathcal{O}(l^d)$.

Cada servidor S_{α_j} con $j \in [k']$ devuelve la operación XOR de los índices que ha recibido. Además, simula a todos los servidores cuyo subíndice está a distancia 1 de $(\alpha_{j1}, \alpha_{j2}, \ldots, \alpha_{jd}) \in \mathbb{F}_2^d$. Por cada servidor que simula manda l bits y como en total simula d servidores, cada S_{α_j} manda $dl+1$ bits. En total todos los servidores participantes mandan $k'(dl+1) \in \mathcal{O}(k'dn^{\frac{1}{d}})$ bits.

Sumando todos los órdenes, la complejidad de comunicación total es $\mathcal{O}(k'dn^{\frac{1}{d}})$. □

Observación 2.20. Por la demostración del anterior teorema, para que un servidor simule a otro cuyos subíndices están a distancia de Hamming 1 del suyo, necesita enviar l bits. Si en vez de simular a servidores cuyo subíndice esté a distancia 1, simulase a aquellos cuyo subíndice esté a distancia t se enviarían l^t bits por cada servidor simulado. De esta manera se cubren todos los posibles subconjuntos que hubiese recibido el servidor a distancia t y consecuentemente los bits que hubiese devuelto. En este caso la comunicación total del protocolo no sería $\mathcal{O}(k'dn^{\frac{1}{d}})$ sino $\mathcal{O}(k'dn^{\frac{t}{d}})$. Por lo que no se consideran códigos de recubrimiento de radio mayor a 1, ya que la comunicación total es mayor.

Observación 2.21. La funcionalidad del anterior protocolo recae en la capacidad de encontrar códigos de recubrimiento de radio 1 en $\mathbb{Z}_2^d$ con el menor número de palabras. De esta manera recubriremos el número total de servidores, k, con el menor número de servidores participantes, k'. Con esto la complejidad de comunicación disminuirá al disminuir el valor k'. Veamos esto en la siguiente subsección.

2.2.1. Número de palabras de un código de recubrimiento de $\mathbb{Z}_2^d$ con radio 1

Sea $C \subseteq \mathbb{Z}_2^d$ un código de recubrimiento no necesariamente lineal de radio 1. Cada bola de radio 1 contiene $d+1$ elementos de $\mathbb{Z}_2^d$, la palabra del centro correspondiente a un servidor participante y los que están a distancia 1, que como tiene d componentes son d. Como el espacio completo tiene 2^d elementos y cada bola tiene $d+1$, el número de palabras que tiene que tener el código, k', satisface $k' \geq \frac{2^d}{d+1}$. Este límite inferior no se alcanza siempre. Lo que buscamos en esta subsección es alcanzarlo para que el número de servidores participantes, k', sea menor y la complejidad total de comunicación del protocolo también. Veamos cuándo se puede conseguir y su relación con los códigos de Hamming.

Definición 2.22. *Una función de recubrimiento es una función $f : \mathbb{F}_2^M \to \mathbb{F}_2^m$ que satisface que para cada $x \in \mathbb{F}_2^M$ e $y \in \mathbb{F}_2^m$ existe algún $x' \in \mathbb{F}_2^M$ tal que $d_H(x, x') \leq \rho$ y $f(x') = y$. El conjunto de funciones de recubrimiento con estos parámetros lo denotamos $COV(\rho, M, m)$.*

Teorema 2.23. *Son equivalentes:*

1. *Existe una función de recubrimiento $COV(\rho, M, m)$, $f : \mathbb{F}_2^M \to \mathbb{F}_2^m$.*
2. *Existe una partición de $\mathbb{F}_2^M$ en 2^m códigos de recubrimiento de radio ρ.*

Demostración. Veamos primero que 1 implica 2. Por la definición de función de cobertura para cada $y \in \mathbb{F}_2^m$, $f^{-1}(y)$ es un código de recubrimiento de $\mathbb{F}_2^M$, porque para cada $x \in \mathbb{F}_2^M$ e $y \in \mathbb{F}_2^m$ cualesquiera siempre existe un $x' \in \mathbb{F}_2^M$ tal que $d_H(x, x') \leq \rho$ y $f(x') = y$. Entonces, variando los $x \in \mathbb{F}_2^M$ y seleccionando siempre el mismo $y \in \mathbb{F}_2^m$, obtenemos todos los $x' \in \mathbb{F}_2^M$ tales que para cualquier $x \in \mathbb{F}_2^M$, $d_H(x, x') \leq \rho$. El conjunto de todos los x' es un código de recubrimiento de $\mathbb{F}_2^M$, en concreto es $f^{-1}(y)$.

Además, las antiimágenes de distintos $y \in \mathbb{F}_2^m$ son disjuntas por lo que $\mathbb{F}_2^M$ es la unión disjunta de 2^m códigos de recubrimiento.

Veamos ahora que 2 implica 1. Dada una partición de $\mathbb{F}_2^M$ en 2^m códigos de recubrimiento basta definir una función que a cada código de recubrimiento le asigne un elemento $y \in \mathbb{F}_2^m$ diferente. □

Veamos un caso particular del anterior teorema del que deduciremos cuándo el límite inferior, $k' \geq \frac{2^d}{d+1}$, se cumple. Para el siguiente teorema, recordemos [18] que un código q-ario de Hamming de orden t, denotado, $Ham_q(t)$, es aquel cuya matriz de control, H_t, tiene por columnas todos los vectores de $\mathbb{F}_q^t$ no proporcionales. En nuestro caso como trabajamos con códigos binarios, $q = 2$ y las columnas de H_t son todos los vectores diferentes de $\mathbb{F}_2^t$ sin añadir el nulo. Estos códigos corrigen errores de peso de Hamming hasta 1. Para encontrar el error se multiplica el vector erróneo por la matriz de control, H_t. Es bien conocido que por tener un error de peso 1 el resultado de este producto, denominado síndrome, será una de las columnas de H_t, precisamente la que corresponde con la coordenada del error. Posteriormente se busca la posición del síndrome en las columnas de la matriz H_t. Finalmente se cambia la coordenada en dicha posición al vector erróneo obteniéndose el vector del código.

Teorema 2.24. *Para cada m existe una función de recubrimiento $COV(1, 2^m - 1, m)$, $f : \mathbb{F}_2^{2^m-1} \to \mathbb{F}_2^m$.*

Demostración. La construcción de esta función es la siguiente. Se escribe una matriz H cuyas columnas sean todos los vectores de $\mathbb{F}_2^m$, excepto el vector nulo. En total habrá $2^m - 1$ columnas. Sea f la función lineal aquella cuya matriz asociada es H. Es decir, por la definición de un código de Hamming, la matriz que acabamos de definir es la matriz de control de un código de Hamming 2-ario de orden m, H_m.

Esta función es de recubrimiento porque para cada $x \in \mathbb{F}_2^{2^m-1}$, $f(x) = y$ es una columna de la matriz construida. Si tomamos otro $y' \in \mathbb{F}_2^m$ podemos encontrar $x' \in \mathbb{F}_2^{2^m-1}$ tal que $d_H(x, x') = 1$ y $f(x') = y'$. Veamos cómo calular x'. Conociendo que $d_H(x, x') = 1$, $x + x'$ sobre $\mathbb{Z}_2$ es un vector de peso de Hamming 1 ya que entre ellos solo cambia una coordenada. El síndrome del vector $x+x'$ es precisamente $(x+x')\cdot H = f(x+x') = f(x)+f(x') = y+y'$. Así, mediante la decodificación de errores del código de Hamming, para determinar la única posición en que se diferencian x y x' basta encontrar la posición del vector $y + y'$ en la matriz H. □

Del teorema anterior se deduce la relación con los códigos de Hamming binarios. En concreto, la matriz definida para la función de recubrimiento $COV(1, 2^m - 1, m)$ es la matriz de control de un código de Hamming binario de orden m.

Observación 2.25. Por el Teorema 2.23, dar la función de recubrimiento $COV(1, 2^m - 1, m)$ es lo mismo que dar una partición de $\mathbb{F}_2^{2^m-1}$ en 2^m códigos de recubrimiento de radio 1. Se deduce que hay códigos de recubrimiento disjuntos para $d = 2^m - 1$. De esto podemos deducir cuándo se cumple el límite inferior $k' \geq \frac{2^d}{d+1}$, que será cuando exista dicha partición. Podemos elegir, como ejemplos, $m = 2$ de lo que se deduce $d = 2^m - 1 = 3$ y entonces $k' = \frac{2^d}{d+1} = 2$. Otro ejemplo, $m = 3$, $d = 2^m - 1 = 7$ y como consecuencia $k' = 16$. Entonces, el número de palabras de cada código de recubrimiento es k'. Y $k' = \frac{2^d}{d+1}$ cuando $d = 2^m - 1$, con $m \in \mathbb{N}$.

Ejemplo 2.26. Veamos el caso $2^m - 1 = 3$, de donde se deduce que $m = 2$. Construyamos la función $f : \mathbb{F}_2^3 \to \mathbb{F}_2^2$ de recubrimiento. Se escribe la matriz H, cuyas columnas son todos los vectores de $\mathbb{F}_2^3$ excepto el nulo.

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

Calculemos las antiimágenes de los vectores de $\mathbb{F}_2^2$:
$f^{-1}(0,0) = \{(0,0,0), (1,1,1)\}$
$f^{-1}(1,0) = \{(0,1,0), (1,0,1)\}$
$f^{-1}(0,1) = \{(1,0,0), (0,1,1)\}$
$f^{-1}(1,1) = \{(0,0,1), (1,1,0)\}$
Se ve que cada una de las antiimágenes es un código de recubrimiento en $\mathbb{F}_2^3$, el primero de ellos es lineal y los demás no. Además, con la unión de todas se obtiene $\mathbb{F}_2^3$.

Observación 2.27. Para el caso en que se alcance la cota $k' = \frac{2^d}{d+1}$, como es en el visto en el ejemplo anterior, la complejidad total del protocolo estudiada en el Teorema 2.2 que es $\mathcal{O}(k'dn^{\frac{1}{d}})$ se aminora. Esta reducción de comunicación se obtiene despejando d de $k' = \frac{2^d}{d+1}$, $k = 2^d$ y también de $n \in \mathcal{O}(l^d)$. Sustituyéndolo en la complejidad anterior obtenemos $\mathcal{O}(k'dn^{\frac{1}{d}}) = \mathcal{O}(k' \log_2 k n^{\frac{1}{\log_2 k' + \log_2 \log_2 k}})$.

Observación 2.28. En la anterior observación se ha dejado la complejidad de comunicación en función de k y k' para que posteriormente, sea más fácil compararla con las complejidades obtenidas en otros protocolos, ya que éstas estarán en función de k. Pero, a la hora de poner el protocolo en práctica y obtener el valor de d óptimo para obtener la menor complejidad, es interesante tener la complejidad total en función de una variable. Esto lo podemos conseguir sustituyendo $k' = \frac{2^d}{d+1}$ en $\mathcal{O}(k'dn^{\frac{1}{d}})$, obteniendo $\mathcal{O}(\frac{2^d}{d+1}dn^{\frac{1}{d}})$. Entonces, en el momento de utilizar el protocolo, en que n será conocido, se puede obtener d minimizando la función $\frac{2^d}{d+1}dn^{\frac{1}{d}}$. Por lo tanto, el valor óptimo de d depende del tamaño de la base de datos, n. Una vez obtenido el valor d, se deduce el de $k = 2^d$, $k' = \frac{2^d}{d+1}$ y $l = \lceil \sqrt[d]{n} \rceil$ y se puede aplicar el protocolo.

Observación 2.29. En el apéndice A he desarrollado un ejemplo de este protocolo. Se ha escrito en un apéndice por falta de espacio, pero lejos de su ubicación en el documento es interesante su lectura para entender correctamente el protocolo.

2.3. PIR de Ambainis

En 1997, A. Ambainis [2] obtuvo un nuevo protocolo con una comunicación total menor, reduciendo el exponente de n, es decir, el segundo factor que se exponía en la Observación 2.4. Para conseguirlo, se combinaron dos protocolos, uno para dos servidores y otro para $k-1$ servidores, consiguiendo uno para k servidores. En este trabajo, primero se detallará el protocolo para $k = 2$ servidores y por recursividad se obtendrá el general para k servidores.

Tanto en el protocolo de $k = 2$ servidores como en su generalización a k servidores, vamos a tener un número de servidores totales y una parte de ellos serán participantes, es decir, recibirán consultas y devolverán respuestas a U. El número de servidores participantes es k y simularán a los no participantes.

Para la redacción de esta sección también me he basado en [10].

2.3.1. Protocolo para $k = 2$ servidores.

Primero veamos el esquema para dos servidores, es decir, $k = 2$. Sin embargo, no sustuiremos su valor, porque posteriormente para obtener el protocolo general para k servidores se hará por recursividad. En ese razonamiento, será útil tener las complejidades totales de comunicación en función de k.

Observación 2.30. Es importante resaltar que aunque aquí también tenemos un número reducido de servidores que simulan al resto, como en el protocolo anterior, no se asignan los servidores a simular mediante un código de recubrimiento, pues el número de servidores simulados no tiene por qué ser el mismo para todos los servidores participantes.

En este protocolo, el número total de servidores será 2^{2k-1} que los asociaremos con tuplas de $\mathbb{F}_2^{2k-1}$, escribiendo sus subíndices en base 2. Sólo $k = 2$ serán participantes, es decir, recibirán consultas de U y responderán a las mismas simulando al total de servidores. El motivo de la elección de $2k-1$ se verá posteriormente en la Observación 2.42. Para el correcto funcionamiento del protocolo, como los subíndices de los servidores en base 2 tienen $2k-1$ componentes, los subíndices de los bits también se tienen que

escribir como una tupla de longitud $2k-1$. Entonces, la base en la que se tienen que escribir es $l = \lceil \sqrt[2k-1]{n} \rceil$ y por lo tanto $n \in \mathcal{O}(l^{2k-1})$. Así pues se escribe cada índice $j \in \mathbb{Z}_n$ de los bits de la base de datos en base l, $(j_1, \ldots, j_{2k-1}) \in \mathbb{Z}_l^{2k-1}$.

Análogamente, podemos denotar a cada servidor como S_α, con $\alpha \in \mathbb{Z}_k$ o como $S_{(\alpha_1,\ldots,\alpha_{2k-1})}$ cuyo subíndice es una tupla en base 2. De manera análoga podremos denotar a cada bit como x_j con $j \in \mathbb{Z}_n$ o como $x_{(j_1,\ldots,j_{2k-1})}$ cuyo subíndice es una tupla en base l.

Veamos el paso a paso del protocolo:

1. U quiere obtener x_i, dónde $i \in \mathbb{Z}_n$, que en base l será $(i_1, \ldots, i_{2k-1}) \in \mathbb{Z}_l^{2k-1}$. U escoge $2k-1$ subconjuntos aleatorios e independientes $W_1^0, W_2^0, \ldots, W_{2k-1}^0 \subseteq \mathbb{Z}_l$ y calcula $W_j^1 = W_j^0 \oplus \{i_j\}$, $\forall j \in [2k-1]$. U envía W_j^0, $\forall j \in [2k-1]$ al servidor $S_{0^{2k-1)}_0}$ y W_j^1, $\forall j \in [2k-1]$ al servidor $S_{1^{2k-1)}_1}$.

   Veamos las respuestas que dan los dos servidores participantes $S_{0^{2k-1)}_0}$ y $S_{1^{2k-1)}_1}$. Entre ambos tienen que simular a los 2^{2k-1} servidores totales.

2. Las respuestas que da el servidor $S_{0^{2k-1)}_0}$ a U son:

   - Primero, realiza la operación XOR de todos los índices recibidos:

   $$\bigoplus_{j_1 \in W_1^0, \ldots, j_{2k-1} \in W_{2k-1}^0} x_{j_1,\ldots,j_{2k-1}}$$

   Este bit que devuelve es el correspondiente al servidor $S_{0^{2k-1)}_0}$, es decir, él mismo.

   - Además, el servidor $S_{0^{2k-1)}_0}$ va a simular a todos los servidores $S_{\beta_1 \ldots \beta_{2k-1}}$ con $(\beta_1, \ldots, \beta_{2k-1}) \in \mathbb{F}_2^{2k-1}$ tales que $d_H((0, \overset{2k-1)}{\ldots}, 0), (\beta_1, \ldots, \beta_{2k-1})) = 1$. La forma de realizarlo es la misma que la explicada en el punto 2 del PIR basado en códigos de recubrimiento. Como $S_{0^{2k-1)}_0}$ no conoce el índice $(i_1, \ldots, i_{2k-1}) \in \mathbb{Z}_l^{2k-1}$ del bit que desea el usuario no puede saber cuáles son los subconjuntos W_j^1, $\forall j \in [2k-1]$.

   Entonces, para simular a cada uno va a devolver todos los posibles bits que hubiese devuelto dicho servidor. Por lo tanto, realiza la operación XOR de todos los posibles subconjuntos $\overline{W}_1^0, \ldots, \overline{W}_{2k-1}^0$ que cumplan lo siguiente: para uno y sólo un índice $h \in [2k-1]$, $\overline{W}_h^0 = W_h^0 \oplus t$ con $t \in \mathbb{Z}_l$. Y para el resto de índices $j \in [2k-1] \setminus \{h\}$, $\overline{W}_j^0 = W_j^0$.

   $$\bigoplus_{j_1 \in \overline{W}_1^0, \ldots, j_{2k-1} \in \overline{W}_{2k-1}^0} x_{j_1,\ldots,j_{2k-1}}$$

   Es decir, con esta respuesta el servidor $S_{0^{2k-1)}_0}$ simula a todos los servidores cuyo subíndice esté a distancia de Hamming 1 de él, realizando para cada uno de ellos l veces la operación XOR.

   Por lo tanto, con el servidor $S_{0^{2k-1)}_0}$ se ha obtenido la respuesta de dicho servidor y todas las posibles respuestas de todos los servidores cuyo subíndice está a distancia 1 de $(0, \overset{2k-1)}{\ldots}, 0)$. Es decir, faltaría la respuesta de los servidores

cuyo subíndice tiene peso de Hamming mayor o igual a 2. En total, éstos son $2^{2k-1} - 2k$ servidores, que es el total menos los que tienen peso de Hamming 1, que son $2k-1$, y el único de peso 0.

3. Las respuestas del servidor $S_{1^{2k-1)}_1}$ a U son:

   - Primero, realiza la operación XOR de todos los índices recibidos:

   $$\bigoplus_{j_1 \in W_1^1, \ldots, j_{2k-1} \in W_{2k-1}^1} x_{j_1,\ldots,j_{2k-1}}$$

   Este bit que devuelve es el correspondiente al servidor $S_{1^{2k-1)}_1}$, es decir, él mismo.

   - Además, el servidor $S_{1^{2k-1)}_1}$ va a simular a todos los servidores $S_{\beta_1 \ldots \beta_{2k-1}}$ con $(\beta_1, \ldots, \beta_{2k-1}) \in \mathbb{F}_2^{2k-1}$ que el servidor $S_{0^{2k-1)}_0}$ no ha simulado. Serán aquellos que cumplan que $1 < d_H((0, \overset{2k-1)}{\ldots}, 0), (\beta_1, \ldots, \beta_{2k-1})) \leq 2k-1$ o lo que es lo mismo $d_H((1, \overset{2k-1)}{\ldots}, 1), (\beta_1, \ldots, \beta_{2k-1})) \leq 2k-3$. Como $S_{1^{2k-1)}_1}$ no conoce el índice $(i_1, \ldots, i_{2k-1}) \in \mathbb{F}_2^{2k-1}$ del bit que desea el usuario no puede conseguir los subconjuntos W_j^0, $\forall j \in [2k-1]$. Entonces, para simular a los servidores $S_{\beta_1 \ldots \beta_{2k-1}}$ tendrá que realizar la operación XOR con todas las combinaciones posibles, como se explica a continuación.

     Realiza la operación XOR de todos los posibles $\overline{W}_1^1, \ldots, \overline{W}_{2k-1}^1$ que cumplan lo siguiente: para dos índices $h, m \in [2k-1]$, $\overline{W}_h^1 = W_h^1$ y $\overline{W}_m^1 = W_m^1$ y para el resto de $j \in [2k-1] \setminus \{h, m\}$, $\overline{W}_j^1 = W_j^1 \oplus t$ con $t \in \mathbb{Z}_l$ ó $\overline{W}_j^1 = W_j^1$

     $$\bigoplus_{j_1 \in \overline{W}_1^1, \ldots, j_{2k-1} \in \overline{W}_{2k-1}^1} x_{j_1,\ldots,j_{2k-1}}$$

     Como $S_{1^{2k-1)}_1}$ simula a todos los servidores $S_{\beta_1 \ldots \beta_{2k-1}}$ con $(\beta_1, \ldots, \beta_{2k-1}) \in \mathbb{F}_2^{2k-1}$ tales que $d_H((0, \overset{2k-1)}{\ldots}, 0), (\beta_1, \ldots, \beta_{2k-1})) \leq 2k-3$, simula a todos los servidores cuyo subíndice en base 2 tienen peso de Hamming como mínimo 2. Es decir, con la respuesta de $S_{1^{2k-1)}_1}$ se terminan simulando los $2^{2k-1} - 2k$ servidores que faltaban.

4. De todos los bits recibidos correspondientes a los servidores simulados, U sólo hará XOR de los que corresponden a los índices del bit deseado.

   Claramente, $S_{0^{2k-1)}_0}$ y $S_{1^{2k-1)}_1}$ devuelven más bits de los que necesita el usuario. Como este envío lo hacen en orden, U sabe qué bits son los que simulan a los servidores y son los que recoge. Finalmente hace la operación XOR y obtiene el bit que desea.

Observación 2.31. Para el caso de $k = 2$ que estamos estudiando, cuando el servidor $S_{1^{2k-1)}_1}$ simule a aquellos cuyo subíndice esté a distancia menor o igual que $2k-3$ (esto se realiza en el paso 3), sustituyendo $k = 2$ serán los que estén a distancia 1 que son pocos servidores. Entonces, el coste de comunicación total no será tan excesivo como a

simple vista podría parecer. Pero cuando k sea un número más grande tendrá que simular a muchos servidores y en comparación con el servidor $S_{0^{2k-1)}0}$ mandará a U muchos más bits. Esta idea se usará posteriormente para aplicar la recurrencia.

Observación 2.32. Analicemos la relación del procedimiento del protocolo con las aplicaciones formales de un protocolo PIR descritas en la Definición 2.1.

Las aplicaciones Q_j con $j \in \mathbb{Z}_k$, que en este caso serían $Q_{0^{2k-1)}0}$ y $Q_{1^{2k-1)}1}$. En estas aplicaciones se utilizan los conjuntos aleatorios W_j^0 con $j \in [2k-1]$. La imagen de $Q_{0^{2k-1)}0}$ que es W_j^0, $\forall j \in [2k-1]$ y la de $Q_{1^{2k-1)}1}$ que es W_j^1, $\forall j \in [2k-1]$.

Las aplicaciones A_j con $j \in Z_k$, que en este caso serían $A_{0^{2k-1)}0}$ y $A_{1^{2k-1)}1}$ correspondientes a las respuestas devueltas a U por cada uno de los servidores.

Finalmente, la aplicación R queda reflejada en el paso 3, como U conoce el índice de interés, los conjuntos aleatorios y las respuestas de todos los servidores que se han devuelto en orden, hace la operación XOR con los bits adecuados, obteniendo el bit deseado.

Observación 2.33. La relación del procedimiento del protocolo con las aplicaciones formales de un protocolo PIR descritas en la Definición 2.1, se haría de manera análoga a la realizada en la Observación 2.9 para el primer protocolo.

Proposición 2.34. *En este protocolo se cumplen las propiedades de privacidad y corrección.*

Demostración. La propiedad de privacidad se cumple. Cada servidor recibe $2k-1$ subconjuntos de $\mathbb{Z}_l$ aleatorios e independientes y de éstos no puede obtener ninguna información del índice de interés de U.

La propiedad de corrección se cumple. De todos los bits que mandan los servidores, U sólo hará XOR de los que corresponden a los 2^{2k-1} servidores, estos bits son:

$$\bigoplus_{j_1 \in \overline{W}_1, \ldots, j_{2k-1} \in \overline{W}_{2k-1}} x_{j_1,\ldots,j_{2k-1}}$$

Donde $\overline{W}_1, \ldots, \overline{W}_{2k-1}$ cumplen que ó $\overline{W}_j = W_j^0$ ó $\overline{W}_j = W_j^1$, $\forall j \in [2k-1]$. Estas combinaciones son bits ya mandados por alguno de los dos servidores, por las siguientes razones:

- Si $\overline{W}_j = W_j^1$ para sólo un $j \in [2k-1]$ o para ninguno, entonces es un bit de los mandados por el primer servidor. Éste hace XOR de $\overline{W}_1^0, \ldots, \overline{W}_{2k-1}^0$, donde todos los subconjuntos son igual a W_j^0 con $j \in [2k-1]$, excepto para un índice $h \in [2k-1]$, que se toma $\overline{W}_h^0 = W_h^0 \oplus t$ para algún $t \in \mathbb{Z}_l$ esto será igual a W_j^1. Además también hace XOR para el caso de que todos sean W_i^0, $i \in [2k-1]$.
- Si $\overline{W}_j = W_j^1$ para dos o más $j \in [2k-1]$. Entonces es un bit de los mandados por el segundo servidor, porque el segundo hace XOR de todos los posibles $\overline{W}_1^1, \ldots, \overline{W}_{2k-1}^1$ con la condición de que dos de ellos, por ejemplo $\overline{W}_h^1$ y $\overline{W}_m^1$ sean igual a W_h^1 y W_m^1 respectivamente, para $h, m \in [2k-1]$. El resto son igual a W_j^1 o a $W_j^1 \oplus t$ para $j \in [2k-1] \backslash \{h, m\}$ y $t \in \mathbb{Z}_l$, en el caso $W_j^1 \oplus t$ estarán incluidos los W_i^0 para algún $t \in \mathbb{Z}_l$. Además también hace XOR para el caso de que todos sean W_j^1, $j \in [2k-1]$.

Es decir, U termina haciendo XOR de los bits $x_{j_1,\dots,j_{2k-1}}$ tales que $j_1 \in W_1^{\alpha_1}, \dots, j_{2k-1} \in W_{2k-1}^{\alpha_{2k-1}}$, $\forall(\alpha_1, \dots, \alpha_{2k-1}) \in \mathbb{F}_2^{2k-1}$. Esos bits son los correspondientes a las respuestas que hubiesen mandado los 2^{2k-1} servidores que, son las estudiadas en el protocolo de la sección 2.1 para $d = 2k-1$. Por ello la corrección de este esquema se deduce de la corrección de aquel explicada en la Proposición 2.1. El bit $x_{i_1\dots i_{2k-1}}$ sólo se encuentra una vez. El resto de bits están en un número par de veces y se cancelan dos a dos. □

Proposición 2.35. *La complejidad de comunicación de este protocolo para $k = 2$ es del orden $\mathcal{O}(n^{\frac{1}{3}})$.*

Demostración. Veamos el número total de bits intercambiados entre el usuario y los servidores.

- U manda a cada servidor $2k-1$ subconjuntos de $\mathbb{Z}_l$. En total, como hay dos servidores participantes y usando que $n \in \mathcal{O}(l^{2k-1})$ se obtiene $2(2k-1)l = \mathcal{O}(kn^{\frac{1}{2k-1}})$.
- El servidor $S_{0^{2k-1)}_{\dots}0}$ devuelve el bit que le corresponde y adicionalmente tantos bits como servidores simula, es decir, tantos como combinaciones haya de $\overline{W}_1^0, \dots, \overline{W}_{2k-1}^0$ cumpliendo las propiedades mencionadas. En cada combinación, $\overline{W}_j^0 = W_j^0$, $\forall j \in [2k-1]\backslash\{h\}$, es decir, se mantienen todos los subconjuntos como se recibieron excepto uno. A éste se le hace la operación XOR para todos los posibles l elementos de $\mathbb{Z}_l$. Es decir, hay $2k-1$ subconjuntos y a cada uno se le realizan l veces la operación XOR. En total el servidor $S_{0^{2k-1)}_{\dots}0}$ manda $(2k-1)l+1 \in \mathcal{O}(kn^{\frac{1}{2k-1}})$ bits.
- El servidor $S_{1^{2k-1)}_{\dots}1}$ devuelve el bit que le corresponde y adicionalmente tantos bits como servidores simula, es decir, tantos como combinaciones haya de $\overline{W}_1^1, \dots, \overline{W}_{2k-1}^1$ cumpliendo las propiedades mencionadas. Para cada combinación existen dos índices $h, m \in [2k-1]$ tales que $\overline{W}_h^1 = W_h^1$ y $\overline{W}_m^1 = W_m^1$. El resto pueden ser W_j^1 o $W_j^1 \oplus t$ para $j \in [2k-1]\backslash\{k, h\}$ y $t \in \mathbb{Z}_l$. Se simulan $2^{2k-1} - 2k$ servidores como se detalló al final del paso 2 del protocolo.

  Por cada servidor que simula hay como mucho l^{2k-3} posibilidades de elegir los subconjuntos $\overline{W}_1^1, \dots, \overline{W}_{2k-1}^1$. Debido a que como máximo hay $(2k-3)$ subconjuntos para los que hay que elegir $t \in \mathbb{Z}_l$. Por tanto, como mucho hay $(2^{2k-1} - 2k)l^{2k-3}$ posibilidades de tomar los $\overline{W}_1^1, \dots, \overline{W}_{2k-1}^1$. En total, como máximo el servidor $S_{1^{2k-1)}_{\dots}1}$ manda $(2^{2k-1} - 2k)l^{2k-3} + 1$ bits, es decir, $\mathcal{O}(2^{2k}n^{\frac{2k-3}{2k-1}})$.

Sumando todos los órdenes de complejidad de comunicación obtenemos que el orden total es $\mathcal{O}(2^{2k}n^{\frac{2k-3}{2k-1}})$. Si sustituimos $k = 2$ la complejidad es de $\mathcal{O}(16n^{\frac{1}{3}}) = \mathcal{O}(n^{\frac{1}{3}})$ □

Observación 2.36. He desarrollado un ejemplo de este esquema para $k = 2$ servidores en el Apéndice B. He considerado escribirlo en un apédice porque en la mayoría de aspectos por la elección de $k = 2$ es muy similar al Ejemplo A.1.

2.3.2. Construcción del protocolo para k servidores

Como se ha explicado antes, al final del protocolo U realiza la operación XOR de los bits $x_{j_1,\dots,j_{2k-1}}$ tales que $j_1 \in W_1^{\alpha_1}, \dots, j_{2k-1} \in W_{2k-1}^{\alpha_{2k-1}}$, $\forall(\alpha_1, \dots, \alpha_{2k-1}) \in \mathbb{F}_2^{2k-1}$. Observemos que de todos los $(2^{2k-1} - 2k)l^{2k-3} + 1$ bits que devuelve el servidor $S_{1\overset{2k-1)}{\dots}1}$, U sólo necesita $2^{2k-1} - 2k$, que son los correspondientes a los bits que hubiesen mandado los servidores cuyo subíndice en base 2 tiene peso de Hamming mayor o igual a 2. Es decir, la mayor parte de la comunicación va del servidor $S_{1\overset{2k-1)}{\dots}1}$ al usuario.

Partiendo de este razonamiento aplicaremos recursividad al protocolo para recuperar de manera más eficiente los bits que se necesitan de la respuesta del servidor $S_{1\overset{2k-1)}{\dots}1}$. Así se obtiene un protocolo general para k servidores con menor comunicación.

Esta construcción se va a realizar a partir de un protocolo de 2 servidores y otro de $k-1$ servidores. A su vez, el de $k-1$ servidores se obtendrá del mismo modo por recursividad. Ya se ha visto el protocolo para dos servidores, $S_{0\overset{2k-1)}{\dots}0}$ y $S_{1\overset{2k-1)}{\dots}1}$. Veamos el procedimiento para k servidores, $S_0, \dots, S_{k-1}$, los denotamos así con los subíndices $\alpha \in \mathbb{Z}_k$ para diferenciar su notación de los servidores del protocolo de $k = 2$, $S_{0\overset{2k-1)}{\dots}0}$ y $S_{1\overset{2k-1)}{\dots}1}$:

1. Los subconjuntos que se mandaban al servidor $S_{0\overset{2k-1)}{\dots}0}$ se mandan al servidor S_0. Los que se mandaban al servidor $S_{1\overset{2k-1)}{\dots}1}$ ahora se mandan a todos los servidores $S_1, \dots, S_{k-1}$.

2. El servidor S_0 manda las mismas respuestas que $S_{0\overset{2k-1)}{\dots}0}$ en el protocolo de $k = 2$ servidores.

3. Los servidores $S_1, \dots, S_{k-1}$ reciben la consulta de U y simulan la respuesta del servidor $S_{1\overset{2k-1)}{\dots}1}$ en el protocolo de $k = 2$, aunque no mandan la respuesta a U.

4. U calcula la longitud de la respuesta que devolverían, la denotamos m.La puede calcular porque conoce el número y el orden de los bits que devolvería S_{111}, y por tanto en este caso $S_1, \dots, S_{k-1}$. Además, tiene en cuenta las posiciones de los $2^{2k-1} - 2k$ bits que necesita de dicha respuesta, $0 \leq m_1, \dots, m_r \leq m - 1$, con $j = 2^{2k-1} - 2k$ que sabe en qué posiciones están porque se devuelven en orden.

5. A continuación, U va a imitar el protocolo para el caso de $k-1$ servidores para obtener los $m_1, \dots, m_r$ bits, de la respuesta de longitud m que hubiesen mandado los servidores $S_1, \dots, S_{k-1}$. Como en el paso anterior los $k-1$ servidores han generado la misma respuesta , todos comparten la misma base de datos. De esa nueva base de datos U quiere acceder a los bits con índices $m_1, \dots, m_r$. Por tanto, de forma análoga al paso 1, envía r consultas de la misma base de datos a los $k-1$ servidores. Las consultas que hubiese mandado a $S_{0\overset{2k-1)}{\dots}0}$ se las manda a S_1 y las que hubiese mandado a $S_{1\overset{2k-1)}{\dots}1}$ a $S_2, \dots, S_{k-1}$. A estas consultas los $k-1$ servidores, $S_1, \dots, S_{k-1}$, contestan imitando el protocolo para $k-1$ servidores. S_1 manda la respuesta que hubiese mandado $S_{0\overset{2k-1)}{\dots}0}$ y $S_2, \dots, S_{k-1}$ operan la respuesta que hubiese mandado $S_{1\overset{2k-1)}{\dots}1}$ en el protocolo de $k = 2$ servidores, pero no la devuelven a U. De nuevo como en el paso 4 U calcula el tamaño de esta respuesta y los bits de ésta que necesita. Esta será la nueva base de datos utilizada para repetir el protocolo para $k-2$ servidores.

6. Los pasos 1-4 se repiten recursivamente cada vez con un servidor menos hasta llegar a dos servidores que devuelven las respuestas explicadas en la subsección anterior.

7. U recopila recursivamente los bits que le interesan y hace un XOR con ellos para obtener el bit deseado de la base de datos inicial.

Ejemplo 2.37. Veamos un ejemplo de cómo se realiza esta recursividad. Supongamos que tenemos 5 servidores participantes, $S_0, \ldots, S_4$, que tienen que simular los bits que hubiesen enviado un total de $2^{2k-1} \overset{k=5}{=} 512$ servidores.

Inicialmente el servidor S_0 recibiría la consulta que U hubiese enviado a S_{000} y los servidores $S_1, \ldots, S_4$ la correspondiente a S_{111}. S_0 imita a S_{000} y devuelve la respuesta que hubiese dado S_{000}, con ella envía los bits que hubiesen devuelto $2k \overset{k=5}{=} 10$ servidores. Los servidores $S_1, \ldots, S_4$ imitan a S_{111}, pero no devuelven dicha respuesta porque conlleva el envío de $(2^{2k-1} - 2k)l^{2k-3} + 1 \overset{k=5}{=} 502 \cdot l^7 + 1$ bits (el valor l se desconoce porque no se está realizando el protocolo) y U no necesita todos los bits de la misma. Almacenan todos esos bits de la respuesta como una nueva base de datos.

De esta respuesta se desean recuperar $2^{2k-1} - 2k \overset{k=5}{=} 502$ bits que son los correspondientes que hubiesen mandado los 502 servidores que simulan. Entonces, se aplica la recursividad a los servidores $S_1, \ldots, S_4$, $k = 4$, con la nueva base de datos que es la respuesta que hubiesen devuelto de longitud $m = 502 \cdot l^7 + 1$, un total de $2^{2k-1} - 2k \overset{k=5}{=} 502$ veces. Ahora S_1 recibe la consulta múltiple que hubiese recibido S_{000} (sobre la nueva base de datos) y $S_2, \ldots, S_4$ las correspondientes a S_{111}. En total, cada servidor $S_1, \ldots, S_4$ recibe 502 consultas ya que es el número de bits de la nueva base de datos que se quieren recuperar. S_1 devuelve la respuesta imitando la que daría S_{000}. Y los servidores $S_2, \ldots, S_4$ imitan a S_{111}, pero no devuelven dicha respuesta, sino que vuelven a almacenarla para usarla como nueva base de datos en el siguiente paso de recursividad.

Se aplica recursivamente el protocolo a las nuevas bases de datos cada vez con un servidor menos. Esto se repite hasta llegar al caso de dos servidores, S_3 y S_4, que desarrollan por completo el protocolo para $k = 2$ servidores. Es decir, en cada paso se han ido decrementando los servidores que hacían el rol del servidor S_{111} hasta que lo hiciese sólo uno, así se reduce la comunicación total.

Proposición 2.38. *En este protocolo se cumplen las propiedades de privacidad y corrección.*

Demostración. Se cumplen las propiedades de privacidad y corrección ya que el protocolo para k servidores se obtiene por recursividad del de 2 servidores. Cada una de las múltiples consultas que se hacen en cada paso de recursividad se hace del mismo modo que una consulta en el protocolo para dos servidores, por lo que en cada consulta está asegurada la corrección y la privacidada al estarlo en el de $k = 2$ servidores. □

Teorema 2.39. *Existe un esquema PIR con seguridad incondicional para k servidores con comunicación total $\mathcal{O}(2^{k^2} n^{\frac{1}{2k-1}})$*

Demostración. Veámoslo por inducción sobre k.

En el protocolo para $k = 2$ como se vio en la Proposición 2.35, U envía en total $\mathcal{O}(kn^{\frac{1}{2k-1}})$ bits a los dos servidores, el servidor $S_{0^{2k-1)}0}$ devuelve $\mathcal{O}(kn^{\frac{1}{2k-1}})$ bits y $S_{1^{2k-1)}1}$

devuelve $\mathcal{O}(2^{2k}n^{\frac{2k-3}{2k-1}})$ bits. Como, en el proceso de recursión, la respuesta del servidor $S_{1^{2k-1)}_1}$ sólo se envía en el caso de $k=2$ servidores podemos sustituir en su complejidad el numerador por $k=2$ obteniendo el siguiente orden $\mathcal{O}(2^{2k}n^{\frac{1}{2k-1}})$, en el resto del orden no se sustituye porque es interesante tenerlo en dependencia de k para sumar las complejidades. Por lo visto ahora, el protocolo para $k=2$ tiene orden de complejidad $\mathcal{O}(2^{2k}n^{\frac{1}{2k-1}})$.

Por lo tanto, ahora veamos el protocolo general para k servidores, $S_0,\dots,S_{k-1}$.

Inicialmente U manda W_j^0, $\forall j\in[2k-1]$ al servidor S_0 y W_j^1, $\forall j\in[2k-1]$ a los servidores $S_1,\dots,S_{k-1}$. En total, utilizando que $n\in\mathcal{O}(l^{2k-1})$ son $k(2k-1)l\in\mathcal{O}(k^2n^{\frac{1}{2k-1}})$ bits.

La respuesta del servidor S_0 es la misma que la de $S_{0^{2k-1)}_0}$ en el caso de $k=2$ servidores y ya se ha visto que tiene $\mathcal{O}(kn^{\frac{1}{2k-1}})$ bits.

Los servidores $S_1,\dots,S_{k-1}$ simulan al servidor $S_{1^{2k-1)}_1}$, pero esta respuesta no la mandan por lo que no se añaden bits de comunicación. Se usa como la nueva base de datos sobre la que se aplica el protocolo para $k-1$ servidores. Como se ha visto en la Proposición 2.35 la nueva base de datos tiene complejidad $\mathcal{O}(2^{2k}n^{\frac{2k-3}{2k-1}})$ (la respuesta que hubiese dado $S_{1^{2k-1)}_1}$).

Para $k=2$ el protocolo tiene orden de complejidad de $\mathcal{O}(2^{2k}n^{\frac{1}{2k-1}})$, por inducción para $k-1$ servidores tendrá orden $\mathcal{O}(2^{2k}n^{\frac{1}{2k-3}})$. Entonces, aplicamos el protocolo para $k-1$ servidores, con una base de datos de tamaño $\mathcal{O}(2^{2k}n^{\frac{2k-3}{2k-1}})$ y obtenemos un orden de comunicación igual a $\mathcal{O}\left(2^{2k}(n^{\frac{2k-3}{2k-1}})^{\frac{1}{2k-3}}\right)=\mathcal{O}(2^{2k}n^{\frac{1}{2k-1}})$.

Este protocolo para $k-1$ servidores se aplica $2^{2k-1}-2k$ veces ya que ese es el número de bits que queremos recuperar de la nueva base de datos. A su vez, cada base de datos resultante de cada una de las $2^{2k-1}-2k$ veces que se aplica el protocolo para $k-1$, es la nueva base de datos sobre la que se aplica el protocolo para $k-2$ servidores que se aplicará $2^{2k-3}-2k+2\in\mathcal{O}(2^{2k-1}-2k)$ veces. Esto se repite recursivamente hasta llegar al caso de $k=2$ servidores. En total, el número de veces que se aplican los protocolos en la recursividad es del orden $\mathcal{O}((2^{2k-1}-2k)^k)=\mathcal{O}((2^{2k})^k)=\mathcal{O}(2^{k^2})$. Por lo tanto, en el protocolo para k servidores la respuesta de los $S_1,\dots,S_{k-1}$ tendría orden $\mathcal{O}(2^{k^2}n^{\frac{1}{2k-1}})$.

Sumando todas las complejidades se obtiene que hay un protocolo para k servidores con comunicación total $\mathcal{O}(2^{k^2}n^{\frac{1}{2k-1}})$. □

Observación 2.40. La generalización obtenida para k servidores parece que se implanta mediante varias rondas de PIR. Primero consultando a k servidores y recibiendo las respuestas, luego a $k-1$ servidores y recibiendo respuestas, así hasta llegar a 2 servidores. No es así, porque las consultas que manda para $k-1$ servidores no dependen de la respuesta que devuelvan a U, de hecho no devuelven ninguna respuesta los $k-1$ servidores. U sabe como formular las nuevas consultas por el orden en el que devolverían los bits los $k-1$ servidores. Por ello manda todas las consultas juntas al comenzar el protocolo, es decir, las correspondientes a k servidores, $k-1$ servidores, así hasta 2 servidores. Posteriormente, obtendrá las respuestas todas seguidas también. Por lo que el protocolo para k servidores se implanta en una ronda.

En la definición de PIR del principio de la sección se afirmó que aunque un servidor reciba varias consultas, sólo utiliza la última recibida para devolver la respuesta. Esto se ve en este protocolo de k servidores dónde se utiliza la recursividad. Aunque los servidores

reciben varias consultas sólo utilizan la última que reciben para devolver la respuesta a U. El resto se utilizan para formar respuestas pero que no se devuelven a U.

Ejemplo 2.41. Si tuviésemos 5 servidores, $S_0, \ldots, S_4$, el servidor S_1 recibiría dos consultas. La correspondiente al protocolo para 5 servidores y también la de 4 servidores. Con la primera formuló una respuesta que no se mandó a U, simplemente se usó como la nueva base de datos. Pero la que utiliza para mandar la respuesta es la segunda consulta recibida ya que en ese momento imitaría al servidor $S_{0^{2k-1)}_0}$ que sí devuelve respuesta.

Observación 2.42. La complejidad obtenida para k servidores es $\mathcal{O}(2^{k^2} n^{\frac{1}{2k-1}})$. Se podría llegar a pensar que aumentando la longitud de las tuplas de los servidores en base 2, que es $2k-1$, se mejoraría la comunicación total. Es decir, si fuese, por ejemplo, $2k$ se podría llegar a alcanzar una complejidad de $\mathcal{O}(2^{k^2} n^{\frac{1}{2k}})$ la cual sería mejor. Veamos por qué no se consigue este orden.

En la Proposición 2.35 se ha visto que la respuesta de la base de datos $S_{1^{2k-1)}_1}$ tiene orden $\mathcal{O}(2^{2k} n^{\frac{2k-3}{2k-1}})$. Para $k = 2$ es lo mismo que $\mathcal{O}(n^{\frac{1}{3}})$ porque $2k - 3 = 1$. Si los servidores en base 2 tuviesen dimensión $2k$ este orden de complejidad sería $\mathcal{O}(2^{2k} n^{\frac{2k-2}{2k}})$ y sustituyendo $k = 2$ sería $\mathcal{O}(2^{2k} n^{\frac{2}{2k}}) = \mathcal{O}(n^{\frac{1}{2}})$ lo cual empeora la comunicación total en vez de mejorarla. Es decir la complejidad sería $\mathcal{O}(2^{k^2} n^{\frac{1}{k}})$ no la esperada de $\mathcal{O}(2^{k^2} n^{\frac{1}{2k}})$. Lo mismo ocurre para cualquier dimensión mayor que $2k-1$.

Por lo tanto la mayor dimensión que podemos tomar es $2k-1$, obteniendo el menor orden de complejidad posible.

Observación 2.43. En el momento de poner el protocolo en práctica se conoce el número total de bits de la base de datos, n, faltaría fijar k. Como se desea que la comunicación total sea la menor posible, la manera de fijar k sería obteniendo el valor k que minimiza el valor de la complejidad de comunicación, $2^{k^2} n^{\frac{1}{2k-1}}$. Una vez obtenido k, se calcula $l = \lceil \sqrt[2k-1]{n} \rceil$ y se puede utilizar el protocolo.

Capítulo 3

PIR de Beimel et al. basado en polinomios

En el último protocolo del capítulo anterior se alcanzó una complejidad de comunicación total del orden $\mathcal{O}\left(2^{k^2} n^{\frac{1}{2k-1}}\right)$. Varios autores como Y. Ishai, E. Kushilevitz [14] y T. Itoh [15] intentaron mejorar la comunicación total pero sólo consiguieron reducir el factor que mutiplica a $n^{\frac{1}{2k-1}}$. No fue hasta 2002 que A. Beimel, Y. Ishai, E. Kushilevitz y J-F. Raymond [4], consiguieron reducir el segundo factor, $n^{\frac{1}{2k-1}}$, lo cuál es más determinante ya que n es un número muy grande, así consiguieron romper la barrera de complejidad de comunicación del anterior protocolo, $\mathcal{O}(n^{\frac{1}{2k-1}})$.

Por el gran avance que supuso he considerado adecuado presentar este protocolo en un capítulo nuevo. Además, su funcionamiento es completamente diferente a los tres anteriores. Se abandona la notación en base l de los índices de los bits y en base 2 de los índices de los servidores. Para su redacción también me he basado en [10].

Observación 3.1. Para este protocolo renombraremos los subíndices de los n bits de la base de datos, en el capítulo anterior los denotábamos $x = x_0 \dots x_{n-1}$ y en este $x = x_1 \dots x_n$. De la misma manera ocurre con los k servidores, renombramos los índices los servidores de $S_0, \dots, S_{k-1}$ a $S_1, \dots, S_k$. Un motivo es que múltiples veces se usarán productorios, sumatorios y subíndices de 0 a $n-1$ y es más cómodo y menos complicado escribir de 1 a n, igualmente con los subíndices de los k servidores. Otro motivo es el abandono de la notación como tuplas de los subíndices de bits y servidores, que era el principal motivo de usar la anterior notación.

Para desarrollar este protocolo se codifica la base de datos, $x = x_1 \dots x_n$, con un polinomio multivariante, $P_x(Z_1, \dots, Z_m)$, sobre $\mathbb{F}_2$. Este polinomio tendrá m variables y grado d. Además deberá verificar la siguiente condición.

$$\forall i \in [n],\ P_x(E_i) = x_i \tag{3.1}$$

donde E_i, con $i \in [n]$, es el i-ésimo elemento de $\mathbb{F}_2^m$ que tiene peso de Hamming d y diferente a los anteriores. Estos vectores existen si se cumple que $\binom{m}{d} \geq n$, es decir, si la cantidad de vectores de $\mathbb{F}_2^m$ que tienen d unos es mayor o igual que n, porque mínimo se necesitan n vectores con esas carácterísticas que serán los E_i, con $i \in [n]$. De la condición

$\binom{m}{d} \geq n$ se deduce que $m \in \mathcal{O}(n^{\frac{1}{d}})$. Este último razonamiento se obtiene de:

$$n \leq \binom{m}{d} = \frac{m!}{d!(m-d)!} = \frac{m \cdot (m-1) \cdot \ldots \cdot (m-d+1)}{d!} \leq \frac{m^d + \epsilon(m)}{d!}$$

donde se deduce $m \in \mathcal{O}(n^{\frac{1}{d}})$.

Definimos P_x de la siguiente manera para que verifique (3.1).

$$P_x(Z_1, \ldots, Z_m) := \sum_{i=1}^{n} x_i \prod_{E_{i_j}=1} Z_j$$

donde E_{i_j} es la coordenada j-ésima del vector E_i con $i \in [n]$ y $j \in [m]$. Como el peso de Hamming de $E_i \in \mathbb{F}_2^m$ es d el polinomio tendrá grado d. Además, se verifica lo requerido en (3.1) ya que $\prod_{E_{i_j}=1} Z_j$ sólo tomará el valor 1 en caso de que las variables $(Z_1, \ldots, Z_m)$ tomen los valores de las coordenadas E_i y 0 en cualquier otro caso.

Observación 3.2. En esta sección se usarán polinomios en varias variables. Se denotará con letras mayúsculas a las variables de los polinomios que no han sido evaluadas. Y con la misma notación pero en letras minúsculas las que sí han sido evaluadas, porque ciertas variables se evaluarán y otras no, de ahí la distinción.

Observación 3.3. El polinomio P_x está determinado por la base de datos x. Por lo tanto, sólo es conocido por los servidores. U no lo posee porque en tal caso ya tendría el bit x_i que desea.

Este protocolo PIR se basa en evaluar $P_x(E_i)$, manteniendo en secreto el vector E_i a todos los servidores. U genera E_i para $i \in [n]$ y elige aleatoriamente $y_1, \ldots, y_{k-1} \in \mathbb{F}_2^m$. Luego obtiene $y_k \in \mathbb{F}_2^m$ de manera que $\sum_{j=1}^{k} y_j = E_i$. Manda a cada servidor S_j el conjunto $\{y_1, \ldots, y_k\} \setminus y_j$.

Para conseguir evaluar el polinomio P_x en E_i, lo evaluaremos en $\{y_1, \ldots, y_k\}$, entonces es necesario redefinir el polinomio P_x que tiene m variables a otro cuyo valor sea el mismo pero con km variables. Entonces, definimos cada variable de P_x como una suma de k variables, es decir, para cada $h \in [m]$, $Z_h = \sum_{j=1}^{k} Y_{j,h}$. Con la introducción de estas nuevas variables, podemos reescribir P_x, lo denotaremos Q_x.

$$Q_x(Y_{j,h_{j\in[k],h\in[m]}}) = P_x(\sum_{j=1}^{k} Y_{j,1}, \ldots, \sum_{j=1}^{k} Y_{j,m})$$

Y a partir de ahora utilizaremos la notación $Y_j = (Y_{j,1}, Y_{j,2}, \ldots, Y_{j,m})$.

Observación 3.4. Q_x depende de mk variables y sigue teniendo grado d, debido a que los monomios de P_x son del tipo $x_r \cdot Z_{r_1} \cdot Z_{r_2} \cdot \ldots \cdot Z_{r_d}$, donde los índices $r_1, \ldots, r_d$ corresponden con las posiciones de E_r que son 1, para $r \in [n]$. Sustituyendo $Z_h = \sum_{j=1}^{k} Y_{j,h}$, $\forall h \in [m]$ se tiene $x_r \cdot \left(\sum_{j=1}^{k} Y_{j,r_1}\right) \cdot \left(\sum_{j=1}^{k} Y_{j,r_2}\right) \cdot \ldots \cdot \left(\sum_{j=1}^{k} Y_{j,r_d}\right)$. Si se desarrolla la anterior expresión se obtienen los monomios de Q_x, que siguen siendo de grado d.

3.1. Asignación de los monomios de Q_x a los servidores

Al inicio del protocolo, se asigna a cada servidor los monomios de Q_x que va a evaluar. Por los vectores $\{y_1, \ldots, y_k\} \setminus y_j$ mandados a cada servidor S_j, $\forall j \in [k]$, cada variable será conocida por todos los servidores menos uno. Por ejemplo, la variable $Y_{t,s}$ con $t \in [k]$ y $s \in [m]$, será conocida por todos los servidores excepto por el S_t. De esta idea se deduce cómo asignar los monomios de Q_x a los servidores.

Cada monomio de Q_x será encomendado al servidor que mejor lo pueda evaluar. Con esto nos referimos al servidor que conozca más valores de sus variables y pueda devolver el monomio con el menor número de incógnitas. Consideremos un monomio, M, de Q_x que tiene grado d. Por lo visto antes, sólo un servidor se pierde cada variable. Es decir, del grado del polinomio depende la capacidad de los servidores para evaluar sus monomios. Por ejemplo, si el grado es $d = k-3$ siempre existirá algún servidor para cada monomio que pueda evaluarlo por completo ya que cada variable es conocida por todos los servidores menos uno. Como el grado de cada monomio M es menor que k, se puede escoger un servidor que conozca todas las $k-3$ variables.

En general, existe un servidor que se pierde como mucho $\lfloor \frac{d}{k} \rfloor$ de las variables de M. Asignaremos M al servidor que pueda evaluar el mayor número de variables de M. Si hay más de uno se escoge uno aleatoriamente. Veamos distintos casos según el valor de d:

- Para $d = k-1$. Entonces, $\lfloor \frac{d}{k} \rfloor = \lfloor \frac{k-1}{k} \rfloor = 0$. Es decir, podemos encontrar un servidor para cada monomio de Q_x tal que no se pierda ninguna variable. Para cada $j_0 \in [k]$ se puede asignar a S_{j_0} todos los monomios tales que ninguna variable de M está en Y_{j_0}. Se verifica que S_{j_0} puede evaluar el polinomio ya que recibe todos los y_j con $j \in [k] \setminus j_0$.
- Para $d = 2k-1$. Entonces, $\lfloor \frac{d}{k} \rfloor = \lfloor \frac{2k-1}{k} \rfloor = 1$. Es decir, podemos encontrar un servidor para cada monomio tal que como mucho se pierda una variable. Para este caso la asignación se realiza de la siguiente manera:
  - Si existe $j_0 \in [k]$ tal que ninguna variable de M está en Y_{j_0}. Entonces se asigna M a S_{j_0}. Esto es como el caso de $d = k-1$.
  - Consideremos que $\forall j \in [k]$ alguna variable de Y_j está en M. Para algún $j_0 \in [k]$ sólo habrá una variable de Y_{j_0} en M. Asignaremos M a S_{j_0}. Este servidor devolvería el valor del monomio con una incógnita, la variable de Y_{j_0} que había en M. No puede darse el caso de que haya 2 variables o más de cada Y_j, $\forall j \in [k]$ en un monomio por la elección de d, ya que $\lfloor \frac{2k-1}{k} \rfloor = 1$. Además, por reducción al absurdo si hubiese dos variables de cada Y_j, $\forall j \in [k]$, M tendría grado $2k > d$, lo que es imposible por la elección del grado de M.

Observación 3.5. La asignación de los monomios se ha dividido en $d = k-1$ y $d = 2k-1$, ya que son casos diferentes. Para el primer caso la complejidad total del protocolo es mayor que la del segundo caso (se verá en los Teoremas 3.10 y 3.11) y podría haberse omitido. Aún así para entender el procedimiento completo he considerado conveniente su inclusión. De esta manera se entiende mejor el protocolo para $d = 2k-1$. Y, a partir de este último, se construye el protocolo final que va a romper la barrera del de Ambainis explicado en el Capítulo 2.3. Además, en el caso $d = k-1$ están incluidos los valores $d < k-1$ y en $d = 2k-1$ los $k \leq d < 2k-1$, pero no se ha escrito de esta manera porque como se

ha mencionado el primer caso se ha añadido para comprender mejor el segundo que con el caso límite $d = k - 1$ es suficiente y el segundo se ha explicado para comprender la recursividad para la cuál sólo necesitaremos el caso $d = 2k - 1$.

Veamos el funcionamiento del protocolo para $d = k - 1$ y $d = 2k - 1$.

1. U escoge aleatoriamente $y_1, \ldots, y_{k-1} \in \mathbb{F}_2^m$ tal que $\sum_{j=1}^{k} y_j = E_i$. Después, se realiza la asignación de los monomios Q_x a los k servidores. Según si $d = k-1$ ó $d = 2k-1$ se tendrá en cuenta una condición o dos, como se ha explicado al comienzo de esta sección. U manda a cada servidor, S_j, el conjunto $\{y_1, \ldots, y_k\} \setminus \{y_j\}$, $\forall j \in [k]$. Es decir, cada servidor tiene todas las variables menos m.

2. Cada servidor contesta con la suma de los monomios que ha recibido sustituyendo las variables que conoce. En el caso $d = k-1$, cada servidor devolverá un bit, porque conoce todas las variables de los monomios que se le han asignado. En el caso de $d = 2k - 1$ cada uno devuelve como máximo un polinomio de m variables y grado 1. Porque como mucho desconoce 1 variable de cada monomio que se le ha asignado y en total desconoce m variables.

3. U suma todos los bits recibidos por los servidores. Si alguna variable está sin sustituir, la sustituye y obtiene el bit deseado.

Observación 3.6. Veamos la relación del funcionamiento del protocolo con las aplicaciones formales de un protocolo PIR descritas en la Definición 2.1.

La aplicaciones consulta son $Q_j : \mathbb{Z}_n \times \mathbb{Z}_2^{(k-1)m} \longrightarrow \mathbb{Z}_2^{(k-1)m}$ para $j \in [k]$, dónde en $\mathbb{Z}_n$ se toma el índice i del bit que se desea recuperar, en $\mathbb{Z}_2^{(k-1)m}$ el vector aleatorio que está formado por $y_1, \ldots, y_{k-1} \in \mathbb{F}_2^m$. La imagen es la consulta que se realiza al servidor S_j que es $\{y_1, \ldots, y_k\} \setminus \{y_j\}$ perteneciente a $\mathbb{Z}_2^{(k-1)m}$.

La aplicación respuesta $A_j : \mathbb{Z}_2^n \times \mathbb{Z}_2^{(k-1)m} \longrightarrow \mathbb{Z}_2^a$ para $j \in [k]$, dónde $\mathbb{Z}_2^n$ representa la base de datos, del conjunto $\mathbb{Z}_2^{(k-1)m}$ se toma la consulta realizada al servidor S_j. Si estamos en el caso $d = k - 1$, $a = 1$ porque cada servidor devuelve un bit y en el caso $d = 2k - 1$, $a = m + 1$ porque cada servidor devuelve un polinomio de grado 1 en m variables

Finalmente, la aplicación $R : \mathbb{Z}_n \times \mathbb{Z}_2^{(k-1)m} \times (\mathbb{Z}_2^a)^k \longrightarrow \mathbb{Z}_2$, dónde en $\mathbb{Z}_n$ se toma el índice del bit deseado, en $\mathbb{Z}_2^{(k-1)m}$ el vector aleatorio definido en la aplicación consulta y en $(\mathbb{Z}_2^a)^k$ las k respuestas recibidas de los servidores. Esta aplicación sustituye las variables que estén sin sustituir de la respuesta de los servidores, las suma y obtiene el bit deseado que pertenece a $\mathbb{Z}_2$.

Teorema 3.7. *En este protocolo se cumple la propiedad de corrección.*

Demostración. El objetivo de U es evaluar $P_x(E_i)$, sin tener acceso a P_x. Todos los monomios de Q_x se han asignado a algún servidor.

Para el caso $d = k - 1$, los servidores devuelven todos los monomios con todas las variables evaluadas. U los suma todos para formar el polinomio Q_x evaluado en $y_1, \ldots, y_k$. Por la restricción, $\sum_{j=1}^{k} y_j = E_i$, el resultado es el mismo que evaluar $P_x(E_i)$ y por la definición esto es x_i.

Para el caso $d = 2k - 1$, cada servidor S_j, con $j \in [k]$, devuelve un polinomio en m variables de grado 1, $P_j(Y_{j,1}, \ldots, Y_{j,m})$. Las variables del polinomio son justamente las m variables que dicho servidor no conoce de Q_x. Como U conoce todos los y_l con $l \in [k]$ puede sustituir los valores $y_{l,h}$ con $h \in [m]$ y sumar los k polinomios recibidos de todos los servidores, obteniendo lo siguiente.

$$\sum_{j=1}^{k} P_j(y_{j,1}, \ldots, y_{j,m}) = P_x(\sum_{j=1}^{k} y_j) = P_x(E_i) = x_i$$

Es decir, en ambos casos U recupera el bit x_i deseado con probabilidad 1. □

Observación 3.8. Para obtener el bit x_i, en ambos casos, U suma todos los monomios que le devuelven los servidores obteniendo de nuevo Q_x. Luego, sustituye las variables que faltan ya que U conoce todas y así obtiene el bit. Por esta razón no tiene sentido asignar un monomio a más de un servidor, porque al realizar posteriormente la suma de todos los monomios devueltos no se obtiene Q_x.

Proposición 3.9. *En este protocolo se cumple la propiedad de privacidad.*

Demostración. La condición de privacidad se cumple en ambos casos $d = k - 1$ y $d = 2k - 1$. Porque cada servidor recibe $k - 1$ vectores de $\mathbb{F}_2^m$ distribuidos uniforme e independientemente del resto. De ellos no puede obtener ninguna información sobre el índice i. □

Veamos la complejidad de comunicación del protocolos para los casos de $d = k - 1$ y $d = 2k - 1$.

Teorema 3.10. *En el caso $d = k - 1$, se obtiene un esquema PIR con complejidad $\mathcal{O}(k^2 n^{\frac{1}{k-1}})$.*

Demostración. Veamos los bits que se mandan entre U y los k servidores. U manda $k - 1$ vectores de $\mathbb{F}_2^m$ a cada servidor. Como cada servidor conoce el valor de todas las variables que se le otorgan en los monomios, devuelve un bit. Utilizando que $m \in \mathcal{O}(n^{\frac{1}{d}})$ la complejidad de comunicación será $k(k-1)m + k \in \mathcal{O}(k^2 n^{\frac{1}{k-1}})$. □

Teorema 3.11. *En el caso $d = 2k - 1$, se obtiene un esquema PIR con complejidad $\mathcal{O}(k^2 n^{\frac{1}{2k-1}})$.*

Demostración. Veamos los bits que se mandan en este caso. U manda $k-1$ vectores de $\mathbb{F}_2^m$ a cada servidor. Pero en este caso cada uno le devuelve un polinomio de m variables y grado 1, es decir $m + 1$ bits. Esto es porque cada servidor puede no conocer 1 variable de cada monomio que se le otorga. Como en total de las km variables que tiene Q_x no conoce m, podría devolver un polinomio en, como mucho, esas m variables. En total se comunican $k(k-1)m + (m+1)k \in \mathcal{O}(k^2 n^{\frac{1}{2k-1}})$ bits. □

Observación 3.12. Por la evolución en la complejidad respecto al grado en los dos anteriores teoremas podemos llegar a pensar que aumentando aún más el valor de d alcanzaríamos complejidades menores. Esto no es así. En efecto, si tomamos $d > 2k - 1$, entonces $\lfloor \frac{d}{k} \rfloor > 2$. En consecuencia, el polinomio P_j que devuelve cada servidor S_j

tendrá las mismas m variables pero de grado 2 o más. Esto se traduce en que cada servidor mandaría $\mathcal{O}(m^d)$ monomios donde d es el grado del polinomio. Para alcanzar la complejidad menor que promete este protocolo tenemos que hacer uso de la recursión y la descomposición de P_x. Esto lo desarrollaremos en la siguiente sección.

Observación 3.13. En la anterior observación se ha afirmado que un polinomio de m variables y grado d tiene $\mathcal{O}(m^d)$ monomios. Esto se deduce de que el número de monomios que hay en un polinomio de m variables y grado d es $\binom{m+d}{d}$ y desarrollando:

$$\binom{m+d}{d} = \frac{m+d!}{d!m!} = \frac{(m+d-1)(m+d-2)\dots(m+1)}{d!} = \frac{m^d+\epsilon(m)}{d!} \overset{m>d}{\in} \mathcal{O}(m^d)$$

Observación 3.14. En el momento de poner el protocolo en práctica se conoce el número total de bits de la base de datos, n, faltaría fijar k. Como se desea que la comunicación total sea la menor posible, la manera de fijar k sería obteniendo el valor k que minimiza el valor de la complejidad de comunicación, $k^2 n^{\frac{1}{d}}$, sustituyendo d por $k-1$ ó por $2k-1$, según el que queramos tomar. Una vez obtenido k, se calcula d y se elige m tal que $\binom{m}{d} \geq n$ y se puede llevar a cabo el protocolo.

Observación 3.15. He desarrollado un ejemplo para $d = k-1$ que por ser sencillo lo he escrito en el Apéndice C. Este ejemplo ayuda para comprender el ejemplo para $d = 2k-1$ desarrollado en el Apéndice D, se ha ubicado ahí por falta de espacio en el docuemento.

3.2. Recursividad

En esta sección se seguirá manteniendo el inicio del protocolo explicado anteriormente. U escoge aleatoriamente $E_i \in \mathbb{F}_2^m$, con peso de Hamming d para $i \in [n]$, cumpliendo $\binom{m}{d} \geq n$. Cambia la asignación de los monomios como se explicará después, pero se mantiene que U elige $y_1, \dots, y_{k-1} \in \mathbb{F}_2^m$. Después calcula y_k de manera que se cumpla la restricción $\sum_{j=1}^{k} y_j = E_i$ y envía cada a cada servidor, S_j, con $j \in [k]$, los vectores $\{y_1, \dots, y_k\} \backslash y_j$.

Veamos cómo se va a realizar la recursión. El objetivo, como antes, es evaluar P_x en E_i. De esta manera U recuperará x_i. Para ello se determinan distintos subconjuntos de servidores, $V \in [k]$, que no serán disjuntos y sus carácterísticas se verán a continuación y se descompone $P_x(Z_1, \dots, Z_m)$ en una suma de distintos polinomios, $P_V(Z_1, \dots, Z_m)$, cada uno de ellos conocido sólo por los servidores en el subconjunto V. U sólo los tendrá que evaluar en el vector E_i para obtener x_i. Es decir, lo que se quiere conseguir es lo siguiente,

$$P_x\left(\sum_{j=1}^{k} y_j\right) = \sum_{V \subseteq [k]} P_V(E_i) = x_i$$

Veamos qué parámetros se van a usar. Para definir los polinomios P_V, nos apoyaremos en los polinomios $\widetilde{P}_V$, finalmente los polinomios P_V serán igual a los $\widetilde{P}_V$ pero evaluados en ciertas variables. Primero, k' es un límite inferior de $|V|$. Es decir, como mínimo habrá k' servidores que conozcan cada $\widetilde{P}_V$, se verá que habrá alguna excepción de $|V| = 1$. λ es

el otro parámetro a tener en cuenta. Cada servidor, S_j, del subconjunto V conoce el valor de todas las variables $\{Y_1, \ldots, Y_k\}$ menos Y_j. Entonces, se define el parámetro λ como el máximo de variables que un servidor no conoce de un monommio M de $\widetilde{P}_V$. Repitiendo este razonamiento para todos los servidores en V, las variables para las cuales todos los servidores conocen su valor son todas menos como mucho $\lambda|V|$.

Inicialmente, los servidores de cada subconjunto V evaluarán esas variables que conocen todos ellos en el polinomio $\widetilde{P}_V$ correspondiente. Por lo tanto, el resultado será el polinomio P_V que tendrá grado como mucho $\lambda|V|$ porque es el número máximo de variables que pueden no conocer todos los servidores de un monomio. Ahora sólo falta que el servidor U los evalúe en E_i para obtener x_i, este paso es el que se realiza recursivamente, porque los polinomios P_V pueden ser de grado 2 o más y si se devuelven directamente a U la comunicación total sería mayor.

Entonces, se utilizará el polinomio P_V que hubiesen devuelto los servidores de cada subconjunto V como la nueva base de datos sobre la que U aplicará de nuevo el protocolo PIR para conocer sólo los coeficientes necesarios. U sabe qué consultas debe formular a los servidores del subconjunto V que conocen cada nueva base de datos (cada P_V) porque esta respuesta está ordenada con un orden prefijado. U sólo quiere recuperar algunos coeficientes, no todos, porque los polinomios P_V se evalúan en E_i, que tiene peso de Hamming d. Entonces, sólo le interesará recuperar aquellos coeficientes cuyas variables al evaluarlas en E_i tengan todas valor 1. En caso contrario, para los monomios con variables Z_j tales que $E_{i_j} = 0$ con $j \in [m]$ ese monomio tendrá valor 0 y no afecta al cómputo para obtener x_i.

Observación 3.16. Notemos que los polinomios P_j definidos en el Teorema 3.7 se evalúan en los puntos y_l con $l \in [k]\backslash\{j\}$ que no tienen porque cumplir la propiedad de tener peso d. Por eso el reto es descomponer P_x en polinomios P_V que se evalúen en E_i, ya que este vector tiene peso d y esto es necesario para que U no tenga que recuperar todos los coeficientes y la complejidad de comunicación sea menor.

Por lo tanto, para cada polinomio P_V que conocen todos los servidores en V se aplicará de nuevo el protocolo. La nueva base de datos serán los coeficientes ordenados de P_V. Como cada P_V está formado por m variables y todos los servidores en conjunto conocen todas menos $\lambda|V|$, entonces la nueva base de datos está formada por $\mathcal{O}(m^{\lambda|V|})$ coeficientes. En total se quieren recuperar 2^d coeficientes, porque cada una de las d variables de E_i que valen 1 pueden estar o no presentes en el monomio. Esto se conseguirá aplicando 2^d veces el protocolo PIR para los $|V|$ servidores que conocen P_V que tiene $\mathcal{O}(m^{\lambda|V|}) = \mathcal{O}(n^{\frac{\lambda|V|}{d}})$ coeficientes, dónde se ha usado que $m \in \mathcal{O}(n^{\frac{1}{d}})$. Así, la recuperación de estos coeficientes se hará por recursividad.

3.3. Construcción de los polinomios P_V para el caso $k = 3$ servidores y $d = 7$.

Primero veamos la construcción de los polinomios P_V para $k = 3$ servidores y $d = 7$, después la generalizaremos a k servidores. Esta explicación es un ejemplo para que posteriormente se comprenda mejor la generalización a k servidores. Se han elegido $k = 3$ y $d = 7$ porque son valores pequeños para los cuáles ya es necesario aplicar la recursividad.

Para este caso $P_x(Z_1, \ldots, Z_m)$ dependerá de m variables y tendrá grado $d = 7$. Como $k = 3$, $Q_x(\{Y_{j,h}\}_{j\in[3],h\in[m]})$ dependerá de $3m$ variables, dónde se ha sustuido $Z_h = \sum_{j=1}^{3} Y_{j,h}$.

P_x es el polinomio que representa la base de datos, tiene tantos sumandos como bits tenga la misma. Para simplificar, construimos los polinomios P_V para P formado por un solo monomio, $P(Z_1, \ldots, Z_m) = Z_1 \cdot Z_2 \cdot \ldots \cdot Z_d$. Para construir los P_V para P_x haríamos lo mismo en el resto de monomios y los sumaríamos. Por la elección de P, se define $Q(\{Y_{j,h}\}_{j\in[k],h\in[m]}) = \left(\sum_{j=1}^{3} Y_{j,1}\right) \cdot \left(\sum_{j=1}^{3} Y_{j,2}\right) \cdot \ldots \cdot \left(\sum_{j=1}^{3} Y_{j,d}\right)$. El polinomio Q depende de $kd = 21$ variables y está formado por $k^d = 3^7$ monomios de la forma $Y_{j_1,1} \ldots Y_{j_d,d}$. Veamos cuantas variables de los monomios de Q conoce cada servidor.

Proposición 3.17. *Para cada monomio M de Q se cumple una de las siguientes opciones.*

1. *Existe un servidor que conoce todas las variables salvo como mucho una.*

2. *Existen dos servidores que conocen todas las variables menos dos y un tercero que conoce todas las variables menos tres.*

Demostración. Como $d = 7$ los monomios M de Q tienen grado 7, es decir, son de la forma $Y_{j_1,1} \cdot \ldots \cdot Y_{j_7,7}$ con $j_l \in [3]$ y $l \in [7]$. Por las diferentes combinaciones de $j_l \in [3]$ con $l \in [7]$ se deduce la proposición.

En el primer caso se contempla que como mucho haya un índice $j_l \in [3]$ con $l \in [7]$ del servidor en cuestión. En este caso conocerá todas menos como mucho una variable.

El caso 2 se refiere cuando hay mínimo dos índices $j_l \in [3]$ con $l \in [7]$ de cada servidor. Como $d = 7$ habrá dos índices de un servidor, dos de otro y del último tres de donde se deduce el segundo caso. □

Dividiremos los monomios de Q según sean ligeros o pesados. Los ligeros son aquellos en que el índice de un servidor aparece como mucho una vez. Por el contrario, si el índice de cada servidor aparece mínimo dos veces es pesado. Para cada monomio, M, consideremos el conjunto $V(M)$ con los índices de los servidores que aparecen como mucho dos veces en M. Por la Proposición 3.17 para los monomios pesados $|V(M)| = 2$.

Observación 3.18. Inicialmente se puede pensar en la definición de cada P_V asignando todos los monomios con $V(M) = V$ y evaluándolos en y_j con $j \notin V$. Por lo que en el caso $|V(M)| = 2$, P_V tendría grado 4 y habría que evaluarlo en y_j con $j \in V$. Estos vectores no tienen por qué tener peso de Hamming d y no se podría aplicar la recursividad. Veamos la definición de P_V de manera que se pueda evaluar en E_i.

Sea M un monomio pesado $M = Y_{j_1,1} \ldots Y_{j_d,d}$, es decir, $|V(M)| = 2$. Definimos:

$$\widetilde{T}(M) := \prod_{j_q \in V(M)} \left(\sum_{j=1}^{3} Y_{j,q}\right) \prod_{j_q \notin V(M)} Y_{j_q,q} \tag{3.2}$$

$$T(M) := \prod_{j_q \in V(M)} Z_q \prod_{j_q \notin V(M)} y_{j_q,q} \tag{3.3}$$

$T(M)$ se ha evaluado en $y_{j_q,q}$ con $j_q \notin V(M)$, por lo que es un monomio de grado 4 en las variables $Z_1 \dots Z_d$. Para las definiciones anteriores se verifica:

$$\widetilde{T}(M)(\{y_j\}_{j\in[3]}) = T(M)(E_i) \tag{3.4}$$

Además, a través de $\widetilde{T}(M)$ se ha definido una relación de equivalencia.

Proposición 3.19. *Dos monomios pesados M, M' están relacionados, $M \equiv M'$, si $\widetilde{T}(M) = \widetilde{T}(M')$. Esto define una relación de equivalencia entre los monomios pesados.*

Demostración. Veamos que es una relación de equivalencia probando que cumple la propiedad reflexiva, simétrica y transitiva. Sean M, M' y M'' tres monomios pesados.

- Reflexiva. Para todo monomio pesado, M, se cumple que $M \equiv M$ ya que $\widetilde{T}(M) = \widetilde{T}(M)$.
- Simétrica. Si $M \equiv M'$ entonces $\widetilde{T}(M) = \widetilde{T}(M')$, por lo que también se cumple que $M' \equiv M$.
- Transitiva. Veamos que si $M \equiv M'$ y $M' \equiv M''$ entonces $M \equiv M''$. Como $M \equiv M'$ entonces $\widetilde{T}(M) = \widetilde{T}(M')$, por $M' \equiv M''$ se verifica que $\widetilde{T}(M') = \widetilde{T}(M'')$. De esto se deduce que $\widetilde{T}(M) = \widetilde{T}(M'')$ y entonces $M \equiv M''$.

□

Ejemplo 3.20. Veamos cómo se calculan $\widetilde{T}(M)$ y $T(M)$. Supongamos el monomio pesado $M_0 = Y_{1,1}Y_{3,2}Y_{1,3}Y_{1,4}Y_{2,5}Y_{3,6}Y_{2,7}$. Entonces, $V(M_0) = \{2, 3\}$ ya que son los índices de los servidores que están dos veces. $\widetilde{T}(M_0) = (\sum_{j=1}^{3} Y_{j,2})(\sum_{j=1}^{3} Y_{j,5})(\sum_{j=1}^{3} Y_{j,6})(\sum_{j=1}^{3} Y_{j,7})Y_{1,1}Y_{1,3}Y_{1,4}$. De la misma manera se calcula $T(M_0)$ pero sustituyendo $\sum_{j=1}^{3} Y_{j,h}$ por Z_h y evaluando $Y_{1,1}, Y_{1,3}, Y_{14}$, $T(M_0) = Z_2Z_5Z_6Z_7y_{1,1}y_{1,3}y_{1,4}$. Otro monomio que puede pertenecer a esta clase de equivalencia es $M_1 = Y_{1,1}Y_{2,2}Y_{1,3}Y_{1,4}Y_{3,5}Y_{3,6}Y_{2,7}$, ya que $\widetilde{T}(M_0) = \widetilde{T}(M_1)$.

Observación 3.21. Fijándonos en todos los monomios pesados, M, que pertenecen a una clase de equivalencia tienen que cumplir que su $V(M)$ sea igual y que las posiciones en que se encuentran las variables del servidor que no pertenece a $V(M)$ sean las mismas. De esto se deduce que para cada subconjunto $V \subseteq [3]$ tal que $|V| = 2$ hay $\binom{7}{4} = 35$ clases de equivalencia, porque $\binom{7}{4}$ es el número de posiciones distintas en las que se pueden escribir las tres variables del servidor que no pertenece a $V(M)$.

Ahora ya podemos definir los polinomios P_V. Para cada subconjunto $V \subseteq [3]$ tal que $|V| = 2$ escogeremos un monomio, M, representante de cada clase de equivalencia. Sea M_V el conjunto de todos los monomios escogidos. Definimos P_V y $\widetilde{P}_V$ de la siguiente manera.

$$P_V(Z_1, \dots, Z_m) := \sum_{M\in M_V} T(M)(\{Z_q\}_{j_q\in V(M)})$$

$$\widetilde{P}_V(\{Y_{j,h}\}_{j\in[k],h\in[m]}) := \sum_{M\in M_V} \widetilde{T}(M)$$

Por su definición, cada $\widetilde{P}_V$ contiene monomios ligeros y pesados. Pero cada monomio pesado está en un sólo $\widetilde{P}_V$. Como el objetivo es descomponer P cuyas variables están sin sustituir, lo haremos a partir de los polinomios $\widetilde{P}_V$ que también tienen todas las variables sin sustituir. Además como los polinomios $\widetilde{P}_V$ están definidos en las variables $\{Y_{j,h}\}_{j\in[k],h\in[m]}$ descompondremos Q que es igual a P pero definido en esas variables. Por lo que queda definir:

$$Q'(\{Y_{j,h}\}_{j\in[k],h\in[m]}) = Q(\{Y_{j,h}\}_{j\in[k],h\in[m]}) - \widetilde{P}_V(\{Y_{j,h}\}_{j\in[k],h\in[m]})$$

Todos los monomios pesados están en los $\widetilde{P}_V$. Entonces, cada monomio M de Q' es un monomio ligero y se lo asignaremos al servidor, S_j con $j \in [k]$, cuyo índice aparece como mucho una vez. La suma de todos los monomios asignados a S_j es el polinomio P_j. S_j lo evaluará en los y_l, $\forall l \in [k]\backslash\{j\}$ y devolverá como mucho un polinomio de m variables y grado 1. Por esto, cada monomio ligero o no está en Q' o está asignado exactamente a un servidor, S_j. Al igual que con la definición de los polinomios P_V, cuando nos refiramos a un polinomio P_j ya estará evaluado en todas las variables que conoce el servidor S_j. Usando la ecuación (3.4) queda la descomposición:

$$P\left(\sum_{j=1}^{k} y_j\right) = \sum_{V\subseteq[3],|V|=2} P_V(E_i) + \sum_{j=1}^{3} P_j(y_j)$$

En este protocolo $k' = 2$ y $\lambda = 2$, porque los polinomios P_V son conocidos para dos servidores. En los monomios pesados es dónde se puede dar el caso de conocer menos variables. Aún así cada servidor que tiene el monomio conoce todas las variables del mismo excepto dos, por ello $\lambda = 2$.

Ejemplo 3.22. Este ejemplo va a ilustrar cómo son los polinomios P_V. Supongamos $V = \{1, 3\}$. Lo siguiente es tomar un monomio, M_i, de cada una de las 35 clases de equivalencia, tal que $V(M_i) = \{2, 3\}$, $\forall i \in [35]$. Escribamos tres de ellos: $M_1 = Y_{2,1}Y_{2,2}Y_{2,3}Y_{1,4}Y_{3,5}Y_{3,6}Y_{1,7}$, $M_2 = Y_{2,1}Y_{2,2}Y_{1,3}Y_{2,4}Y_{3,5}Y_{3,6}Y_{1,7}$ y $M_3 = Y_{2,1}Y_{2,2}Y_{1,3}Y_{3,4}Y_{2,5}Y_{3,6}Y_{1,7}$ como se ve entre ellos varía las posiciones en las que se encuentran las variables del S_2 que es el que no pertenece a V. Faltarían 32 monomios más. Veamos cómo sería P_V para estos monomios. $P_{\{2,3\}}(Z_1, \ldots, Z_m) = Z_4Z_5Z_6Z_7y_{21}y_{22}y_{23} + Z_3Z_5Z_6Z_7y_{21}y_{22}y_{24} + Z_3Z_4Z_6Z_7y_{21}y_{22}y_{25} + \ldots$ donde faltarían 32 monomios más. De todos estos monomios U sólo recuperará con PIR recursivo aquellos coeficientes cuyas variables Z_h con $h \in [m]$ sean todas igual a 1.

Teorema 3.23. *Existe un protocolo PIR para $k = 3$ servidores con complejidad total $\mathcal{O}(n^{\frac{4}{21}})$.*

Demostración. Veamos la comunición de bits que hay en este protocolo. Inicialmente U manda a cada servidor, S_j, $\{y_1, \ldots, y_k\}\backslash\{y_j\}$. Esto son $k(k-1)m \in \mathcal{O}(k^2 n^{\frac{1}{d}})$ bits.

Por la descomposición de P_x cada polinomio P_V tiene grado 4 y es conocido para los servidores en V y cada polinomio P_j tiene grado 1 conocido por el servidor S_j $\forall j \in [k]$.

Los servidores V que poseen P_V no devuelven el polinomio, que posee $\mathcal{O}(m^4)$ coeficientes. Sino que U recupera los coeficientes que quiere usando recursivamente un protocolo PIR para 2 servidores. Este protocolo PIR lo usa tantas veces como coeficientes quiera recuperar. La nueva base de datos serán los coeficientes del polinomio P_V y se

realiza para dos servidores porque son los que lo conocen. Este protocolo PIR es el del Teorema 3.11 que tiene complejidad $\mathcal{O}(k^2 n^{\frac{1}{2k-1}})$. Como la nueva base de datos tiene $\mathcal{O}(m^4)$ bits y se utiliza para $k = 2$ servidores, usando que $m \in \mathcal{O}(n^{\frac{1}{7}})$, tendrá una complejidad de $\mathcal{O}\left(2^2 (m^4)^{\frac{1}{3}}\right) = \mathcal{O}(4n^{\frac{4}{21}})$.

Los polinomios P_j que quedan tienen m variables y grado 1 por lo que se devuelven a U que los evaluará en las variables que faltan. Esto son $m + 1 \in \mathcal{O}(n^{\frac{1}{7}})$ bits.

En total este esquema tiene complejidad de comunicación $\mathcal{O}(4n^{\frac{4}{21}}) = \mathcal{O}(n^{\frac{4}{21}})$. □

Observación 3.24. Si en vez de haber usado este protocolo recursivo, hubiésemos usado el del Teorema 3.11. La comunicación total hubiese sido $\mathcal{O}(n^{\frac{2}{7}})$ ya que $d = 7$ pero $\lfloor \frac{d}{k} \rfloor = 2$ que es lo que queríamos evitar. En este caso los servidores hubiesen devuelto polinomios de m variables y grado 2.

3.4. Construcción de los polinomios P_V para el caso general de k servidores

Esta generalización se va a basar en las ideas de la subsección 3.3. Como antes, partimos del polinomio $P_x(Z_1, \ldots, Z_m)$ de grado d que codifica la base de datos $x = x_1 \ldots x_n$. U escoge aleatoriamente $E_i \in \mathbb{F}_2^m$, con peso de Hamming d para $i \in [n]$ y también $y_1, \ldots, y_k \in \mathbb{F}_2^m$ sujeto a la restricción $\sum_{j=1}^{k} y_j = E_i$, cumpliendo $m \in \mathcal{O}(n^{\frac{1}{d}})$. Posteriormente, cada servidor, S_j con $j \in [k]$, recibirá de U los vectores $\{y_1, \ldots, y_k\} \backslash \{y_j\}$. Por esta elección reformulamos P_x en $Q_x(\{Y_{j,h}\}_{j\in[k],h\in[m]}) = P_x\left(\sum_{j=1}^{k} Y_{j,1}, \ldots, \sum_{j=1}^{k} Y_{j,m}\right)$

El objetivo es descomponer P_x de la siguiente manera para obtener x_i. Ésta tiene que ser de manera que los P_V se evalúen en $E_i \in \mathbb{F}_2^m$.

$$x_i = P_x\left(\sum_{j=1}^{k} y_j\right) = \sum_{V\subseteq[k],|V|\geq k'} P_V(E_i) + \sum_{j=1}^{k} P_j(y_j) \tag{3.5}$$

Los polinomios P_j son conocidos para los servidores S_j. Son de grado 1 en las variables $Y_{j,h}$ con $h \in [m]$. Los polinomios P_V verifican lo siguiente.

Veamos un pequeño repaso de todos los parámetros que se utilizan en el protocolo. El número total de servidores es k, $S_1, \ldots, S_k$. El límite inferior del número de servidores que puede haber en cada subconjunto $V \subseteq [k]$ es k'. Otro parámetros es λ, que es el máximo de variables que no conoce cada uno de los servidores en V del polinomio P_V. Consecuentemente, todos los servidores en V conocen todas las variables del polinomio menos como mucho $\lambda|V|$. Entonces, el polinomio P_V tendrá grado como mucho $\lambda|V|$. Para cada monomio M de Q_x se define el subconjunto $V(M)$ de todos los servidores que aparecen como mucho λ veces en M.

Es importarte resaltar que el número de servidores que se usan en los protocolos PIR del siguiente paso de recursividad es $|V|$. Y las nuevas bases de datos son los polinomios P_V que tienen tamaño $\mathcal{O}(m^{\lambda|V|})$.

Antes de la construcción de los polinomios P_V veamos dos proposiciones. La primera controla el valor máximo de d de manera que se cumpla lo siguiente.

Proposición 3.25. *Sean $\lambda, k' \leq k$ y $d \leq (\lambda+1)k-(\lambda-1)k'+(\lambda-2)$. Sea M un monomio de grado como mucho d en las variables $\{Y_{j,h}\}_{j\in[k],h\in[m]}$. Entonces, o existe un servidor, S_j, que desconoce como mucho una variable de M o $|V(M)| \geq k'$.*

Demostración. Se demuestra por reducción al absurdo. Supongamos que cada S_j desconoce como mínimo dos variables del monomio M. Y también que como mucho $k'-1$ servidores desconocen como mucho λ variables de M. Esto último es equivalente a decir que como mínimo $(k-(k'-1))$ servidores desconocen como mínimo $\lambda+1$ variables de M.

Por lo tanto, el número de variables de M como mínimo tiene que ser $(k-(k'-1))(\lambda+1)+(k'-1)2$, por el número de variables que desconoce cada servidor. Existen $(k-(k'-1))$ servidores que desconocen como mínimo $\lambda+1$ variables y los $(k'-1)$ servidores que quedan cada uno desconoce como mínimo 2.

En cada uno de los razonamientos no puede darse el caso de que dos servidores desconocan variables iguales. Porque cada servidor, S_j, sólo desconcoce $\{Y_{j,h}\}_{h\in[m]}$ que nunca serán las mismas que las del servidor, $S_{j'}$, que desconoce $\{Y_{j',h}\}_{h\in[m]}$. Por lo tanto, como M tiene d variables y por este razonamiento el número mínimo de variables es $(k-(k'-1))(\lambda+1)+(k'-1)2$, se deduce que $(k-(k'-1))(\lambda+1)+(k'-1)2 = k(\lambda+1)-k'(\lambda-1)+\lambda-1 \geq d$ llegando a contradicción. □

Para cada monomio M de Q_x pesado definamos $\widetilde{T}(M)$ y $T(M)$. Las definiciones son análogas a las dadas en las ecuaciones (3.2) y (3.3). La diferencia es que ahora hay k servidores en vez de 3.

$$\widetilde{T}(M) := \prod_{j_q\in V(M)} \left(\sum_{j=1}^{k} Y_{j,q}\right) \prod_{j_q\notin V(M)} Y_{j_q,q} \tag{3.6}$$

$$T(M) := \prod_{j_q\in V(M)} Z_q \prod_{j_q\notin V(M)} y_{j_q,q} \tag{3.7}$$

Con estas definciones se verifica:

$$\widetilde{T}(M)(\{y_j\}_{j\in[k]}) = T(M)(E_i) \tag{3.8}$$

Sean los monomios pesados $M' = Y_{j'_1,1}Y_{j'_2,2}\dots Y_{j'_m,m}$ y $M'' = Y_{j''_1,1}Y_{j''_2,2}\dots Y_{j''_m,m}$. Como ya se ha visto antes a través de $\widetilde{T}(M)$ se define una relación de equivalencia. Dos monomios pesados, M' y M'', están relacionados, $M' \equiv M''$, si $\widetilde{T}(M') = \widetilde{T}(M'')$. Para que esto suceda se tienen que cumplir dos condiciones. La primera que $V(M') = V(M'') = W$. Y la segunda que para cada índice $q \in [m]$ ó j'_q, j''_q están los dos en W ó $j'_q = j''_q$. Esto último es que la misma variable, $Y_{j_q,q}$, aparezca en ambos monomios.

A partir de ahora, denotaremos por $\delta(M)$ al número de variables $Y_{j_q,q}$ en M con $j_q \in V(M)$.

Ejemplo 3.26. Veamos un ejemplo de cómo se calculan $V(M)$ y $\delta(M)$ para M un monomio de $Q_x(\{Y_{j,h}\}_{j\in[k],h\in[m]})$ de grado d. Supongamos $k = 4$, $k' = 3$, $\lambda = 3$ y por la Proposición 3.25, $d \leq 11$. Tomemos $d = 11$. Sea $M_0 = Y_{4,1}Y_{1,2}Y_{2,3}Y_{4,4}Y_{1,5}Y_{4,6}Y_{3,7}Y_{4,8}Y_{3,9}Y_{2,10}Y_{4,11}$. Como $\lambda = 3$, $V(M_0)$ será el conjunto de todos

los servidores que aparecen como mucho 3 veces. Es decir, $V(M) = \{1, 2, 3\}$. Y por lo tanto $\delta(M_0) = 6$.

Veamos otro monomio, $M_1 = Y_{1,1}Y_{1,2}Y_{2,3}Y_{4,4}Y_{1,5}Y_{4,6}Y_{3,7}Y_{2,8}Y_{3,9}Y_{2,10}Y_{4,11}$, entonces $V(M) = \{1, 2, 3, 4\}$ ya que todos los servidores aparecen como mucho 3 veces. Y $\delta(M_1) = 11$.

Proposición 3.27. *Para cada monomio M_1 de $\widetilde{T}(M)$ se verifican las siguientes propiedades.*

1. *$V(M_1) \subseteq V(M)$.*
2. *$\delta(M_1) \leq \delta(M)$.*
3. *Si $V(M_1) = V(M)$ y $\delta(M_1) = \delta(M)$ entonces $M_1 \equiv M$.*
4. *Sea M_2, si $M_2 \equiv M_1$ entonces M_2 también es un monomio de $\widetilde{T}(M)$.*

Demostración. Veamos la demostración de cada una de ellas. Para ello analizemos todos los monomios que hay en $\widetilde{T}(M)$. Según el monomio, M, escogido su $V(M)$ será distinto y las posiciones, $q \in [m]$, donde estén las variables $Y_{j_q,q}$ con $j_q \notin V(M)$ también. Esto determinará los monomios que habrá en $\widetilde{T}(M)$. En él estarán todos los monomios de la forma $Y_{j_1,1}Y_{j_2,2}\dots Y_{j_m,m}$ tales que los índices de los servidores $j_q \notin V(M)$ estén en los mismos $q \in [m]$ que en M. Para el resto de variables el primer subíndice puede ser cualquiera de $[k]$. Las diferencias que puede tener el monomio $M_1 = Y_{j'_1,1}Y_{j'_2,2}\dots Y_{j'_m,m}$ respecto de $M = Y_{j_1,1}Y_{j_2,2}\dots Y_{j_m,m}$ están en los j'_q relativos a las posiciones $q \in [m]$ de M con $j_q \in V(M)$. El cambio es que dichos j'_q pueden ser cualquier $l \in [k]$.

1. Si el cambio en M_1 es de algún índice j_q de manera que los índices de los servidores en $V(M)$ sigan estando como mucho λ veces, entonces $V(M_1) = V(M)$. El resto de cambios en estas variables sólo pueden desencadenar que algún $j_q \in V(M)$ aparezca más de λ veces. Entonces, dicho j_q ya no pertenezca a $V(M_1)$ que en este caso $V(M_1) \subset V(M)$.

2. Se puede dar el caso de $\delta(M_1) < \delta(M)$ si en los cambios de M_1 respecto de M algún índice j'_q es distinto de los de $V(M)$. Puede darse la igualdad si se cambian los j_q de manera que todos ellos sigan perteneciendo a $V(M)$. No puede darse ya que $\delta(M_1) > \delta(M)$ entonces M_1 no sería un monomio de $\widetilde{T}(M)$.

3. Para que $M_1 \equiv M$ se tiene que cumplir que $V(M_1) = V(M) = W$ y que para cada índice $q \in [m]$ ó $j_q, j_{q'}$ están los dos en W ó $j_q = j_{q'}$. Veamos si se cumple la segunda condición. Por la primera condición los $j'_q \notin V(M_1)$ son los mismos que no están en $V(M)$ por lo tanto ya se cumple para los índices $q \in [m]$ tales que $j_q \notin V(M_1)$ que $j_q = j_{q'}$. Como $\delta(M_1) = \delta(M)$ para el resto de $q \in [m]$ sus correspondientes j_q están en W.

4. Sea $M_2 = Y_{j''_1,1}Y_{j''_2,2}\dots Y_{j''_m,m}$. Como M_1 es un monomio de $\widetilde{T}(M)$ entonces los $j_q \notin V(M)$ como mínimo están en los mismos $q \in [m]$ que M. Además, por $M_2 \equiv M_1$ se cumple que $V(M_2) = V(M_1) \subseteq V(M)$. Por lo que para M_2 los $j_q \notin V(M)$ como mínimo están en los mismos $q \in [m]$ que M. Para resto de $q \in [m]$ da igual cual sea su j''_q ya que sea cual sea, M_2 será monomio de $\widetilde{T}(M)$.

Queda así demostrada la proposición. □

Una vez vistas las dos proposiciones anteriores, veamos el teorema para construir los polinomios P_V.

Teorema 3.28. *Sean λ, k', k y d como en la Proposición 3.25. Sea $P_x(Z_1, \ldots, Z_m)$ un polinomio de grado d y $E_i = \sum_{j=1}^{k} y_j$. Entonces, existen polinomios $P_V(Z_1, \ldots, Z_m)$ para cada $V \subseteq [k]$ tal que $|V| \leq k'$ y polinomios $P_j(Y_{j,1}, \ldots, Y_{j,m})$ para $j \in [k]$ tales que:*

1. *Cada polinomio $P_V(Z_1, \ldots, Z_m)$ tiene como máximo grado $\lambda|V|$ y se obtiene a partir de $P_x(Z_1, \ldots, Z_m)$ y $\{y_j\}_{j \notin V}$.*

2. *Cada polinomio $P_j(Y_{j,1}, \ldots, Y_{j,m})$ tiene como máximo grado 1 y se obtiene a partir de $P_x(Z_1, \ldots, Z_m)$ y $\{y'_j\}_{j' \neq j}$*

3. *Con los polinomios anteriores se verifica la condición de la ecuación 3.5.*

Demostración. Como se hizo en el caso de $k = 3$ servidores, basta con demostrarlo para $P_x(Z_1, \ldots, Z_m)$ formado por un solo monomio. Por lo tanto consideremos $P(Z_1, \ldots, Z_m) = Z_1 \ldots Z_d$. Consecuentemente, $Q(\{Y_{j,h}\}_{j \in [k], h \in [m]}) = \left(\sum_{j=1}^{k} Y_{j,1}\right)\left(\sum_{j=1}^{k} Y_{j,2}\right)\ldots\left(\sum_{j=1}^{k} Y_{j,d}\right)$. Por la defición de $Q(\{Y_{j,h}\}_{j \in [k], h \in [m]})$ es un polinomio de km variables, con k^d monomios. Cada monomio es de la forma $Y_{j_1,1} Y_{j_2,2} \ldots Y_{j_d,d}$.

Veamos un algoritmo para construir los polinomios $P_V(Z_1, \ldots, Z_m)$. En él se va a utilizar un polinomio auxiliar $Q'(\{Y_{j,h}\}_{j \in [k], h \in [m]})$. Algunos monomios pueden ser sacados e introducidos repetidamente en Q'.

1. Inicialmente $Q'(\{Y_{j,h}\}_{j \in [k], h \in [m]}) = Q(\{Y_{j,h}\}_{j \in [k], h \in [m]})$. También inicializamos para todo V tal que $|V| \leq k'$, $P_V(Z_1, \ldots, Z_m) = 0$

2. Para los monomios, M, que hay en Q' se calcula $V(M)$. Se escoge el subconjunto V tal que $V = V(M)$ sea de mayor tamaño. Si hay dos se toma uno cualquiera y nunca se vuelve a los $V(M)$ ya tratados. En caso de que $|V| < k'$ se para el algoritmo.

3. Mientras haya algún monomio M tal que $V(M) = V$. De todos esos monomios se escoge el monomio, M, que maximize $\delta(M)$. Y se calcula:

$$P_V(Z_1, \ldots, Z_m) = P_V(Z_1, \ldots, Z_m) + T(M)$$

$$Q'(\{Y_{j,h}\}_{j \in [k], h \in [m]}) = Q'(\{Y_{j,h}\}_{j \in [k], h \in [m]}) - \widetilde{T}(M)$$

4. Volver al paso 2.

Por la construcción de los P_V son del grado deseado ya que los $T(M)$ son de grado como mucho $\lambda|V|$.

Tenemos que ver que el algoritmo para, es decir en algún momento todos los monomios M de $Q'(\{Y_{j,h}\}_{j \in [k], h \in [m]})$ tienen $|V(M)| < k'$. Observemos que si $M_1 \equiv M_2$ en cualquier paso del algoritmo o ambos monomios están en Q' o ninguno está en Q'. Ya que en cada

iteración el polinomio Q' se modifica sustrayendo $\widetilde{T}(M)$ para un monomio M. Por la propiedad 4 de la Proposición 3.27 si M_1 está en $\widetilde{T}(M)$ y $M_1 \equiv M_2$ entonces M_2 también está en $\widetilde{T}(M)$.

Empleando las propiedades de la Proposicion 3.27 y la observación anterior veamos que el algoritmo para. En cada iteración, en el paso 3, se realiza Q' menos $\widetilde{T}(M)$. En este paso se añaden monomios, M', a Q'. Para los monomios que se añaden se pueden dar tres casos que son los estudiados en la Proposicion 3.27.

- Si $V(M') \subseteq V(M)$, el mononomio M' se tratará en futuras iteraciones del algortimo en el paso 2. Porque primero trata los monomios, M, de mayor $V(M)$.
- Si $\delta(M') \leq \delta(M)$, el mononomio M' se tratará en futuras iteraciones del algortimo en el paso 3. Porque primero trata los monomios, M, de mayor $\delta(M)$.
- Si $V(M') = V(M)$ y $\delta(M') = \delta(M)$ por la propiedad 3, $M' \equiv M$. Y como M está en $\widetilde{T}(M)$ también lo estará M' por lo que se restan ambos monomios de Q', eliminándose.

En resumen, aunque se añaden monomios, M', a Q' siempre tienen o menor $V(M')$ o menor $\delta(M')$. Por lo que se tratarán en las futuras iteraciones y siempre se avanza.

Una vez que el algoritmo ha parado, los monomios M' en Q' cumplen que $|V(M')| < k'$. Por la Proposición 3.25 para cada M' existe un servidor, S_j, que se pierde como mucho una variable de $\{Y_{j,h}\}_{h\in[m]}$. Entonces M' se añade al correspondiente $P_j(Y_{j,1}, \ldots, Y_{j,m})$. Por tanto se verifica la ecuación (3.5). □

Veamos la complejidad de comunicación total de este protocolo utilizando la recursividad.

Teorema 3.29. *Asumiendo que existe un protolo PIR, P, con complejidad de comunicación $C_P(n,k)$. Sean $d, \lambda, k' \in \mathbb{N}$ tales que $k' < k$ y $d \leq (\lambda+1)k-(\lambda-1)k'+(\lambda-2)$. Entonces, existe un protocolo PIR, P', con la siguiente complejidad total de comunicación.*

$$C_{P'}(n,k) = \mathcal{O}\left(n^{\frac{1}{d}} + \sum_{l=k'}^{k} \binom{k}{l} C_P(n^{\frac{\lambda l}{d}}, l)\right)$$

Dónde se han eliminado los factores que dependen de k.

Demostración. Veamos por qué se obtiene esta complejidad recursivamente.

U envía $k-1$ subconjuntos de $\mathbb{F}_2^m$ a cada servidor, estas son las consultas. En total $\mathcal{O}(k^2 n^{\frac{1}{d}})$ bits.

Con los bits mandados se obtiene la descomposición de P_x del Teorema 3.28. Los servidores, S_j, con $j \in [k]$ devuelven los polinomios $P_j(Y_{j,1}, \ldots, Y_{j,m})$ de m variables y grado 1. Esto son $\mathcal{O}(kn^{\frac{1}{d}})$ bits.

U tiene que recuperar recursivamente los 2^d coeficientes de cada polinomio P_V. Para recuperar cada coeficiente aplicará un protocolo PIR cuya base de datos es el tamaño del polinomio, es decir, $\mathcal{O}(n^{\frac{\lambda|V|}{d}})$ y el número de servidores $|V|$. Veamos cuántos polinomios P_V hay. El número de servidores en los subconjuntos V está acotado inferiormente por k'. Por lo que podría haber subconjuntos V con $|V| = l$ con $l = \{k', k'+1, \ldots, k\}$. Y para

cada subconjunto de tamaño l habria $\binom{k}{l}$ subconjuntos. De lo que se deduce el segundo sumando.

Sumando todas las complejidades anteriores y eliminando los factores que dependen de k se obtiene el resultado deseado. □

Observación 3.30. Fijémonos que se verifica que $C_P(n^{\frac{\lambda l}{d}}, l) = \mathcal{O}\left(C_P(n^{\frac{\lambda k'}{d}}, k')\right), \forall l \geq k'$. Porque aunque aumentemos l y consecuentemente el tamaño de la base de datos, el número de servidores va a ser mayor y en total el protocolo para $l \geq k'$ tendrá menor complejidad. Por esta razón , se puede reformular el teorema anterior obteniendo una complejidad de $C_{P'}(n,k) = \mathcal{O}\left(n^{\frac{1}{d}} + C_P(n^{\frac{\lambda k'}{d}}, k')\right)$.

Observación 3.31. Aunque para alcanzar esta complejidad hay que hacer uso de varios protcolos PIR en una ronda se puede implementar entero. Ya que los coeficientes que necesita U de los polinomios P_V están determinados por E_i. U sabe en que posiciones estarán estos coeficientes. Entonces, en la primera ronda podría mandar su consulta inicial y además todas las necesarias para recuperar los coeficientes.

Ejemplo 3.32. Veamos cómo obtenemos los protocolos mejorados por recursividad. Para $k = 3$, estableciendo $\lambda = k' = 2$ y $d = 7$ la complejidad total sería $\mathcal{O}(n^{\frac{1}{7}} + C_P(n^{\frac{2\cdot 2}{7}}, 2))$. Usando el protocolo del Teorema 3.11 se obtiene $\mathcal{O}(n^{\frac{4}{21}})$. Si en vez de usar la recursividad hubiésemos usado directamente el protocolo del Teorema 3.11, la complejidad sería de $\mathcal{O}(n^{\frac{1}{5}})$ que es peor. Además, este protocolo ahora lo usamos recursivamente para los siguientes.

Veamos la complejidad para $k = 4$, con $\lambda = 2$, $k' = 3$ y $d = 9$. Sería $\mathcal{O}(n^{\frac{1}{9}} + C_P(n^{\frac{2\cdot 3}{9}}, 3))$. En vez de usar el protcolo del Teorema 3.11, usamos recursivamente el obtenido en el párrafo anterior, obteniendo una comunicación total de $\mathcal{O}(n^{\frac{8}{63}})$.

Veamos por último que escogiendo los parámetros óptimos podemos obtener una complejidad total de $\mathcal{O}\left(n^{\frac{2\log\log_2 k}{k\log_2 k}}\right)$. Para ello necesitamos un lema previo.

Teorema 3.33. *Para cada $i \in \mathbb{N}$, exite un protocolo PIR P_i tal que $C_{P_i}(n,k) = \mathcal{O}(n^{\frac{2}{ik}})$ para $k \geq (i-1)!$.*

Demostración. La demostración la haremos por inducción de i. Para $i = 3$ se verifica por el Teorema 3.11. Ya que $k \geq 2$, por ejemplo $k = 2$, sería $C_{P_3}(n,k) = \mathcal{O}(n^{\frac{1}{3}})$.

Supongamos por inducción que el teorema se verifica para un cierto i. Veamos que también se verifica para $i+1$.

Supongamos $k' = \lceil \frac{k}{i} \rceil \geq (i-1)!$, $\lambda = \lfloor \frac{i}{2} \rfloor$ y $d = (\lambda+1)k - (\lambda-1)k' \leq (\lambda+1)k - (\lambda-1)k' + (\lambda-2)$. De esta manera estamos en las hipótesis del Teorema 3.29 y podemos aplicarlo.

$C_{P_{i+1}}(n,k) = \mathcal{O}\left(n^{\frac{1}{d}} + C_{P_i}(n^{\frac{\lambda k'}{d}}, k')\right) = \mathcal{O}\left(n^{\frac{1}{d}} + n^{\frac{2\lambda}{id}}\right) = \mathcal{O}\left(n^{\frac{2\lambda}{id}}\right)$. El último paso se deduce de que $\lambda \geq \frac{i}{2}$ entonces $\frac{2\lambda}{i} \geq 1$. Faltaría probar que $\frac{2\lambda}{id} \geq \frac{2}{(i+1)k}$.

$$\frac{\lambda}{id} \overset{1)}{\leq} \frac{\lambda}{i((\lambda+1)k - (\lambda-1)\frac{k}{i})} \leq \frac{\lambda}{k(\lambda i + i - \lambda)} \overset{2)}{\leq} \frac{\lambda}{k(\lambda i + \lambda)} = \frac{1}{k(i+1)}$$

Donde en 1) se han usado las definiciones de d y k' y en 2) la de λ, quedando demostrado el teorema. □

Veamos como conseguir los parámetros óptimos.

Corolario 3.34. *Para $k \geq 3$, existe un protocolo PIR tal que $C_P(n,k) = \mathcal{O}\left(n^{\frac{2\log_2\log_2 k}{k\log_2 k}}\right)$.*

Demostración. Escogiendo $i = \left\lceil \frac{\log_2 k}{\log_2\log_2 k} \right\rceil$ se obtiene que $(i-1)! \leq (i-1)^{(i-1)} \leq (\log_2 k)^{\frac{\log_2 k}{\log_2\log_2 k}} \overset{1)}{=} k$.

Para el 1) tomando logaritmos neperianos a ambas partes se obtiene $\ln\left(\log_2 k^{\frac{\log_2 k}{\log_2\log_2 k}}\right) = \frac{\log_2 k}{\log_2\log_2 k}\ln(\log_2 k) = \frac{\frac{\ln k}{\ln 2}}{\frac{\ln\log_2 k}{\ln 2}}\ln(\log_2 k) = \ln k$.

Es decir, estamos en las hipótesis del Teorema 3.29 y el protocolo P puede ejecutar P_i obteniendo $C_{P_i}(n,k) = \mathcal{O}\left(n^{\frac{2\log_2\log_2 k}{k\log_2 k}}\right)$. □

La explicación de por qué estos parámetros son los óptimos para este protocolo se encuentra en el apéndice B de [4].

Capítulo 4

PIR computacional (cPIR)

En la sección anterior se han visto distintos protocolos PIR en los que los servidores tienen capacidad de computación ilimitada. Por ello, para lograr las propiedades de privacidad y correción del protocolo PIR, la base de datos, $x = x_0 \dots x_{n-1}$, tenía que replicarse en varios servidores. Como se mencionó en la introducción, ahora veremos unos protocolos de PIR computacional (cPIR).

La definición de PIR computacional es la misma que la Definición 2.1 de PIR con seguridad incondicional, exceptuando la propiedad de privacidad. La diferencia con el PIR de seguridad incondicional es que los servidores sólo pueden hacer operaciones en tiempo restringido por su capacidad computacional. Por esto la privacidad del protocolo va a residir en problemas sin solución para su límite computacional y así reduciremos la comunicación total. Por este motivo ahora la privacidad está definida en términos computacionales.

Para definir la privacidad se va a introducir el término de circuitos lógicos, [**?**].

Definición 4.1. *Un circuito lógico, C, es una función booleana, dónde las operaciones de suma, producto y tomar complementario definidas en el álgebra de Boole, se implementan con las puertas lógicas OR, AND y NOT.*

En los protocolos cPIR existirá un valor, t, el parámetro de seguridad, que cuantifica el nivel de protección criptográfica que ofrece el protocolo, es decir, determina la robustez del esquema contra intentos de violar la privacidad de las consultas realizadas por el usuario al servidor. De normal se denota por k pero en este documento lo denotaremos t ya que sino se puede confundir con el número de servidores. A medida que t aumenta, la seguridad del protocolo también aumenta, haciendo más difícil para un servidor romper la privacidad de las consultas.

Para explicar la condición de privacidad, el servidor va a utilizar un circuito lógico, C_t, para intentar distinguir dos consultas sobre índices diferentes. El circuito lógico, C_t, modelará las capacidades computacionales del servidor que intenta distinguir las consultas para obtener el índice del bit en el que está interesado el usuario. Su subíndice t indica cuál es su parámetro de seguridad, se toma superior a una cota T que es el mínimo valor del parámetro para que no se rompa la privacidad. Estos circuitos serán de tamaño acotado polinomialmente en n, es decir, los servidores sólo pueden hacer operaciones cuya complejidad sea polinomial en la variable n, que es el número de bits de la base de datos. Los circuitos C_t se aplicarán a $Q(\star, \star)$ que es la aplicación que formaliza las consultas

detallada en la Definición 2.1 y se obtiene un 1 si el servidor averigua que la consulta pertenece a un determinado índice y 0 en caso contrario. En el contexto que estamos $t \ll n$, por ejemplo, se puede tomar $t = n^\alpha$ con $0 < \alpha < 1$. Entonces, se reescribe la propiedad de privacidad para un protocolo cPIR como:

2*. Un modelo cPIR cumple la propiedad de **privacidad** si cumple lo siguiente. Sea la base de datos $x = x_0 \dots x_{n-1}$, $c > 0$ una constante, $\epsilon > 0$ entonces para cualesquiera bits x_j, x_h cuyos índices son $0 \leq j, h \leq n-1$ y para todas las familias de circuitos de tamaño acotado polinomialmente en n, $\{C_t\}_t$, existe $T \in \mathbb{N}$ tal que $\forall t > T$.

$$|P(C_t(Q(j,u)) = 1) - P(C_t(Q(h,u)) = 1)| < \frac{1}{\text{máx}(t,n)^c} < \frac{1}{n^c} < \epsilon$$

donde, $P(C_t(Q(j,u)) = 1)$ mide la probabilidad de que un servidor averigüe que la consulta pertenece a un determinado índice j. Como se quiere que las consultas sean indistinguibles esto se formaliza con que la probabilidades $P(C_t(Q(j,u)) = 1)$ y $P(C_t(Q(h,u)) = 1)$, $\forall j \neq h$ sean prácticamente iguales. Esto asegura que el servidor no puede distinguir entre las consultas con una probabilidad mayor que $\frac{1}{2}$.

De esta manera, el índice i del bit x_i que quiere recuperar U estará oculto de manera computacional a los servidores. Como se verá más adelante, con estas premisas se construirán protocolos cPIR con mejor complejidad que para el caso de PIR con seguridad incondicional.

4.1. Problema de los residuos cuadráticos

El protocolo que se explicará en la siguiente sección se basa en la dificultad de saber si $y \in \mathbb{N}$ es residuo cuadrático módulo n un número compuesto, si no se conoce la factorización del mismo. En esta sección se analizará el por qué de este motivo.

Primero, repasemos unos conceptos previos que se utilizarán en el protocolo [21].

Sea $(G, *)$ un grupo abeliano y denotemos con ϕ al indicador o función de Euler. Un grupo G es cíclico si existe un $g \in G$ tal que es generador del grupo. El orden de un grupo abeliano, Z_n^*, es $\phi(n)$.

En el trabajo estos conceptos se usarán en el caso de $\mathbb{Z}_p^*$, con p primo. Sea $g \in \mathbb{Z}_p^*$, g es generador del grupo si el subgrupo de genera es el total. Para ello, por el teorema de Lagrange el orden de g tiene que ser $p-1$. Como el orden de g es el mínimo m tal que $g^m = 1$, g será generador si $g^r \not\equiv 1$ módulo p para todo divisor r de $p-1$.

Para comprender el siguiente protocolo cPIR se necesitan unas definiciones previas sobre teoría de números [10, 8, 17, 12].

Definición 4.2. *Sean $a, n \in \mathbb{N}$ dos números primos entre sí. Se dice que a es un residuo cuadrático módulo n o simplemente residuo si existe $b \in \mathbb{Z}_n^*$ tal que $b^2 \equiv a \ mod \ n$.*

Observación 4.3. A lo largo del trabajo se usará la notación $Q_n(y) = 0$ si y es un residuo cuadrático módulo n y $Q_n(y) = 1$ en caso contrario.

Definición 4.4. *Sea p un número natural primo y $a \in \mathbb{N}$. El símbolo de Legendre de a módulo p se define de la siguiente manera.*

$$\left(\frac{a}{p}\right) = \begin{cases} 1 & \textit{si } a \textit{ es un residuo mod } p \\ 0 & \textit{si } a \equiv 0 \textit{ mod } p \\ -1 & \textit{si } a \textit{ no es un residuo mod } p \end{cases}$$

Veamos ahora una generalización del símbolo de Legendre para el caso de $n \in \mathbb{N}$ impar y compuesto.

Definición 4.5. *Sea $n \in \mathbb{N}$ un número impar y compuesto cuya factorización es $n = p_1^{\alpha_1} \cdot \ldots \cdot p_r^{\alpha_r}$, con p_i primo y $\alpha_i \in \mathbb{N}$, $\forall i \in \{1, \ldots, r\}$ y $a \in \mathbb{N}$. El símbolo de Jacobi de a módulo n se define de la siguiente manera.*

$$\left(\frac{a}{n}\right) = \left(\frac{a}{p_1}\right)^{\alpha_1} \cdot \left(\frac{a}{p_2}\right)^{\alpha_2} \cdot \ldots \cdot \left(\frac{a}{p_r}\right)^{\alpha_r}$$

donde $\left(\frac{a}{p_i}\right)$ es el símbolo de Lengendre de a módulo p_i, $\forall i \in \{1, \ldots, r\}$.

Observación 4.6. El problema de calcular un residuo cuadrático se considera difícil cuando se realiza módulo un número compuesto, n, difícil de factorizar. Un número compuesto es fácil de factorizar si sus factores están muy próximos o si alguno de ellos es muy pequeño. Esto se conoce gracias al estudio de los algoritmos de factorización de Fermat y de ρ de Pollard, porque éstos son eficientes en dichos casos [20].

Al tratarse de un problema que no se puede resolver bajo limitaciones computacionales lo usaremos para construir el protocolo cPIR. Por ello elegiremos el número compuesto, n, del subconjunto

$$H_t = \{n/n = pq \text{ con } p, q \text{ primos de longitud } \frac{t}{2} \text{ bits y suficientemente separados}\}$$

Analicemos a continuación por qué es difícil conocer si un número es residuo cuadrático módulo un número compuesto, $n \in H_t$ difícil de factorizar.

Veamos que se puede calcular el símbolo de Jacobi $\left(\frac{y}{n}\right)$ sin conocer la descomposición de n. Para ello necesitamos las siguientes propiedades, demostradas en [21] y en concreto una demostración sencilla para la propiedad 4 en [19].

1. Si $y \equiv z$ módulo n, entonces $\left(\frac{y}{n}\right) = \left(\frac{z}{n}\right)$.
2. Sean $z, y \in \mathbb{N}$. El símbolo de Jacobi es una función multiplicativa $\left(\frac{yz}{n}\right) = \left(\frac{y}{n}\right)\left(\frac{z}{n}\right)$.
3. Se tiene que $\left(\frac{-1}{n}\right) = (-1)^{\frac{n-1}{2}}$ y $\left(\frac{2}{n}\right) = (-1)^{\frac{n^2-1}{8}}$.
4. Ley de la reciprocidad cuadrática $\left(\frac{y}{n}\right) = \left(\frac{n}{y}\right)(-1)^{\frac{(n-1)(y-1)}{4}}$.

Algortimo para calcular el símbolo de Jacobi $\left(\frac{y}{n}\right)$. Siempre que se pueda se aplican las propiedades 1 y 2 para simplificar el cálculo. Con lo que obtengamos tras su cómputo se aplica la propiedad 4. Realizar esto recursivamente hasta obtener ó $\left(\frac{-1}{n}\right)$ ó $\left(\frac{2}{n}\right)$. El resultado será 0, 1 ó -1.

Observación 4.7. Este algoritmo explicado para calcular el símbolo de Jacobi se puede aplicar para calcular el símbolo de Legendre. Por lo que saber si un número es residuo cuadrático módulo un número primo será fácil.

Veamos un ejemplo de cómo se calcula el símbolo de Jacobi sin factorizar n.

Ejemplo 4.8. Calculemos el símbolo de Jacobi de $y = 248$ módulo $n = 1313$, sin factorizar 1313. $\left(\frac{248}{1313}\right) = \left(\frac{2}{1313}\right)^3 \left(\frac{31}{1313}\right)$ donde se ha usado la propiedad 2. Al primer factor le aplicamos la fórmula $\left(\frac{2}{1313}\right) = (-1)^{\frac{1313^2-1}{8}} = 1$. Veamos el cáculo del segundo factor: $\left(\frac{31}{1313}\right) \overset{1)}{=} (-1)^{\frac{(30\cdot 1312)}{4}} \left(\frac{1313}{31}\right) \overset{2)}{=} \left(\frac{11}{31}\right) \overset{3)}{=} (-1)\left(\frac{31}{11}\right) \overset{4)}{=} -\left(\frac{9}{11}\right) \overset{5)}{=} -\left(\frac{3}{11}\right)^2 = -1$. Se le ha aplicado la ley de reciprocidad cuadrática en los pasos 1 y 3. En los pasos 2 y 4 se ha aplicado la propiedad 1. En el 5 la propiedad 2.

Por lo que se obtiene $\left(\frac{248}{1313}\right) = 1^3(-1) = -1$

Observación 4.9. Por la definición y las propiedades del símbolo de Jacobi $\left(\frac{y}{n}\right)$, éste puede tener valor -1, 0 y 1. Tendrá valor 0 cuando $y \equiv 0$ módulo n. Además, por la propiedad 1 podemos suponer $y \in \mathbb{Z}_n$.

Veamos la Proposición 4.10 y el Teorema 4.13 para entender el significado de los símbolos de Jacobi -1 y 1. Para demostrar el teorema usaremos dos proposiciones previas.

Proposición 4.10. *Sea n un número compuesto impar e $y \in \mathbb{Z}_n$. Si el símblo de Jacobi $\left(\frac{y}{n}\right) = -1$ entonces y no es un residuo cuadrático módulo n.*

Demostración. La demostración se basa en el Teorema Chino de los Restos. Sea n un número compuesto tal que $n = pq$ con p, q primos. Un entero y será residuo cuadrático módulo n si lo es módulo p y módulo q.

El entero y es residuo cuadrático módulo n si existe $x \in \mathbb{Z}_n^*$ tal que $x^2 \equiv y \bmod n$. Por el Teorema Chino de los Restos, $x^2 \equiv y \bmod n$ si $x_p^2 \equiv y$ módulo p y $x_q^2 \equiv y$ módulo q tales que $x \equiv x_p$ módulo p y $x \equiv x_q$ módulo q. Es decir, y es residuo cuadrático módulo p y módulo q. Por lo tanto, los símbolos de Legendre de y módulo p y módulo q son igual a 1. Entonces el símbolo de Legendre de y módulo n, $\left(\frac{y}{n}\right) = \left(\frac{y}{p}\right)\left(\frac{y}{q}\right) = 1$. Por lo que si es -1 no será residuo cuadrático. □

Proposición 4.11. *[21] Sea $g \in Z_p^*$ un generador del grupo cíclico Z_p^*. Entonces g^s es un residuo cuadrático módulo p si y solo si s es par.*

Demostración. Primero veamos que si s es par entonces $g^s \bmod p$ es residuo cuadrático. Podemos escribir $s = 2k$. De esta manera, $g^s = g^{2k} = (g^k)^2$. Esto demuestra que g^s es residuo cuadrático módulo p, dado que g^s es igual al cuadrado de $g^k \in Z_p^*$ módulo p. Veamos que $g^k \in Z_p^*$, esto se cumple porque g es un generador de Z_p^* y por lo tanto cualquier potencia suya pertenece a Z_p^*.

Veamos que si g^s módulo p es residuo cuadrático, entonces s es par. Como g^s es un residuo cuadrático podemos escribir $x^2 \equiv g^s$ módulo p para $x \in Z_p^*$. Elevamos a ambos lados a $\frac{(p-1)}{2}$. Por el teorema de Euler $x^{(p-1)} \equiv 1$ módulo p. Entonces faltaría ver que $1 \equiv g^{\frac{s(p-1)}{2}}$ módulo p. Como g es un generador de Z_p^*, $g^{\frac{(p-1)}{2}} \not\equiv 1$ módulo p. Luego, para conseguir lo deseado $\frac{s(p-1)}{2}$ debe ser múltiplo de $p-1$, es decir, s tiene que ser par. □

Proposición 4.12. *Sea p un número primo. Entonces, la mitad de los elementos de Z_p^* son residuos cuadráticos módulo p y la otra mitad no.*

Demostración. La demostración se deduce de la proposición anterior. Como g es un generador del grupo cíclico Z_p^*, se cumple que $Z_p^* = \{1, g, g^2, \ldots, g^{(p-2)}\}$, todos ellos módulo p. La mitad de los exponentes de g son pares y la otra mitad impares. Entonces, por la anterior proposición la mitad son residuos cuadráticos y la otra no. □

Teorema 4.13. *Sea $n = pq$ un número compuesto impar con p, q primos. La mitad de los $y \in \mathbb{Z}_n^*$ tienen símbolo de Jacobi -1, la otra 1. De los últimos la mitad son residuos cuadráticos y la otra mitad no.*

Demostración. El símbolo de Jacobi de $y \in \mathbb{Z}_n^*$ módulo n es $\left(\frac{y}{n}\right) = \left(\frac{y}{p}\right)\left(\frac{y}{q}\right)$. Por la proposición anterior sabemos que en $\mathbb{Z}_p^*$ la mitad de elementos son residuos cuadráticos y la otra mitad no, igualmente para $\mathbb{Z}_q^*$. Por lo que se van a dar 4 posibilidades:

$$\left(\frac{y}{p}\right) = 1 \quad \text{y} \quad \left(\frac{y}{q}\right) = 1 \tag{4.1}$$

$$\left(\frac{y}{p}\right) = -1 \quad \text{y} \quad \left(\frac{y}{q}\right) = 1 \tag{4.2}$$

$$\left(\frac{y}{p}\right) = 1 \quad \text{y} \quad \left(\frac{y}{q}\right) = -1 \tag{4.3}$$

$$\left(\frac{y}{p}\right) = -1 \quad \text{y} \quad \left(\frac{y}{q}\right) = -1 \tag{4.4}$$

Las 4 combinaciones son equiprobables porque que $y \in \mathbb{Z}_n^*$ sea residuo cuadrático o no módulo p es independiente de que lo sea o no módulo q.

Si $\left(\frac{y}{n}\right) = -1$ significa que y es residuo cuadrático sólo módulo p ó módulo q. Esto ocurre para la mitad de los $y \in \mathbb{Z}_n^*$ que corresponde a los casos (4.2) y (4.3).

Para la otra mitad de los $y \in \mathbb{Z}_n^*$, $\left(\frac{y}{n}\right) = 1$. De ellos sólos serán residuos cuadráticos los que lo son para p y para q, es decir, los correspondientes al caso (4.1), por lo que sólo ocurre para la mitad de ellos. La otra mitad son no residuos cuadráticos correspondientes a (4.4). □

Observación 4.14. Si al calcular el símbolo de Jacobi de y módulo n obtenemos un 1 no podemos asegurar que y es residuo cuadrático módulo n. Por lo que para saber si $y \in \mathbb{Z}_n$ es residuo cuadrático módulo n compuesto, sería necesario factorizarlo. De aquí se obtiene la idea de escoger un número n no factorizable en tiempo polinómico, para que no se pueda conocer si y es residuo cuadrático módulo n.

Si no se conoce la factorización de n e $\left(\frac{y}{n}\right) = 1$, no hay ningún procedimiento para saber si y es un residuo cuadrático módulo n o no. Este problema fue estudiado por Gauss como uno de los cuatro problemas principales de la aritmética y se conoce como el *Problema de los Residuos Cuadráticos.*

En este caso el parámetro de seguridad es t ya que la seguridad depende de poder factorizar o no n, de t bits, producto de dos primos de longitud $\frac{t}{2}$ bits cada uno. Cuánto

mayor sea t más difícil será factorizar n, entonces se define un umbral T a partir del cuál para $t > T$ la factorización no será posible en tiempo acotado polinómicamente. Normalmente se suele tomar $T = 2048$ bits [3].

Podemos formalizar el problema de residuos cuadráticos de la siguiente manera, dónde el circuito C_t tratará de averiguar si $y \in \mathbb{Z}_n$ es residuo cuadrático módulo n con $n \in H_t$, se utilizará la notación explicada en la Observación 4.3. Por lo tanto $C_t(n, y) = 0$ si el circuito averigua que y es residuo cuadrático módulo n y $C_t(n, y) = 1$ en caso contrario.

Problema de los Residuos Cuadráticos. Para cada constante $c > 0$ y cada familia de circuitos de tamaño polinomial, $\{C_t\}_t$, existe $T \in \mathbb{Z}$ tal que $\forall t > T$

$$P(C_t(n, y) = Q_n(y)) < \frac{1}{2} + \frac{1}{t^c}$$

donde $n \in H_t$ y $\left(\frac{y}{n}\right) = 1$.

Por lo que, la probabilidad de que el servidor usando el circuito, C_t, averigue si $y \in \mathbb{Z}_n$ es residuo cuadrático módulo n es prácticamente $\frac{1}{2}$. Entonces, no existe familia de circuitos que pueda computar si un número es residuo cuadrático de manera más eficiente que el azar.

4.2. cPIR de Kushilevitz y Ostrovsky basado en residuos cuadráticos

Este protocolo fue estudiando por E. Kushilevitz y R. Ostrovsky en 1997 [17] y se basa en el problema de los residuos cuadráticos. Supuso un avance significativo porque obtiene una complejiadd de comunicación menor que las obtenidas en protocolos PIR y además no es necesaria la replicación de la base de datos en varios servidores, es decir, el número de servidores es $k = 1$. En el caso de PIR con seguridad incondicional para un servidor, la comunicación total es de $\mathcal{O}(n)$ bits, porque se devuelve a U la base de datos completa.

4.2.1. Funcionamiento del protocolo

Veamos el funcionamiento del protocolo por pasos. Para este protocolo sólo se necesitará un servidor, es decir, la base de datos no está replicada. Consideremos la base de datos, $x = x_0 \dots x_{n-1}$ de n bits como una matriz $s \times r$, con $sr \in \{n, n+1, \dots, n+s-1\}$, porque no tiene sentido hacerla más grande de lo estrictamente necesario para formar las s filas. Se representa con una matriz $s \times r$, y no $\lceil\sqrt{n}\rceil \times \lceil\sqrt{n}\rceil$, para que el número de filas y columnas sea distinto y se comprenda mejor el protocolo. En el momento de aplicarlo se podrá escoger representarla como una matriz $s \times r$ ó $\lceil\sqrt{n}\rceil \times \lceil\sqrt{n}\rceil$. Como la base de datos está representada en forma de matriz, supongamos que el usuario está interesado en el bit x_i con $i \equiv (i_1, i_2) \in \mathbb{Z}_s \times \mathbb{Z}_r$.

Observación 4.15. Como la base de datos se escribe en una matriz $s \times r$ podríamos escribir las componentes de los índices como elementos de $[s] \times [r]$ en vez de $\mathbb{Z}_s \times \mathbb{Z}_r$. Pero, en la práctica la escritura de la base de datos normalmente se hará en una matriz $\lceil\sqrt{n}\rceil \times \lceil\sqrt{n}\rceil$ y en ese caso las componentes de los índices se escriben en base $l = \lceil\sqrt{n}\rceil$ y pertenecerán a $\mathbb{Z}_l \times \mathbb{Z}_l$, así es más sencillo de ordenar los bits. Además es la notación que se ha usado a lo largo del Capítulo 2 en este trabajo.

Observación 4.16. En este protocolo se requiere que sea imposible en tiempo acotado polinómicamente saber si un número es residuo cuadrático o no. Por ello primero se elegirá un número compuesto $m \in H_t$, para que no se pueda factorizar en tiempo acotado polinómicamente. Se toma el número compuesto m, en vez de n como en la sección anterior, para que no se confunda con el número total de bits de la base de datos que es n. Los y se eligirán de manera que $\left(\frac{y}{m}\right) = 1$. Porque el símbolo de Jacobi es fácil de calcular y si se obtiene un -1 ya se sabría que y no es residuo cuadrático módulo n.

1. Inicialmente, U escoge un número compuesto $m \in H_t$. Por lo que toma dos números primos p_1 y p_2 de longitud $\frac{t}{2}$ tales que $m = p_1 p_2$ no es factorizable por los algoritmos conocidos. También escoge $y_0, y_1, \ldots, y_{r-1} \in \mathbb{Z}_m$ tales que $\left(\frac{y_i}{m}\right) = 1$, $\forall i \in \mathbb{Z}_r$. Como U conoce la descomposición de m en factores primos puede elegir los $y_0, y_1, \ldots, y_{r-1}$ que sean residuos cuadráticos o no.

   Se escogerán de modo que todos ellos serán residuos cuadráticos menos y_{i_2} que no es un residuo cuadrático. U envía al servidor $m, y_0, y_1, \ldots, y_{r-1}$ y mantiene la descomposición de m en secreto.

2. El servidor recibe $y_0, y_1, \ldots, y_{r-1}$, como $\left(\frac{y_i}{m}\right) = 1$, $\forall i \in \mathbb{Z}_r$ no puede conocer si son residuos cuadráticos módulo n o no. El servidor va a realizar los siguientes cálculos.

$$w_{h,j} = \begin{cases} y_j^2 \text{ mód } m & \text{si } x_{h,j} = 0 \\ y_j \text{ mód } m & \text{si } x_{h,j} = 1 \end{cases}$$

   Primero computa los $w_{h,j}$ para $h \in \mathbb{Z}_s$ y $j \in \mathbb{Z}_r$, formando una nueva matriz $\{w_{h,j}\}_{h \in \mathbb{Z}_s, j \in \mathbb{Z}_r}$. Posteriormente, multiplica todos los elementos de la misma fila obteniendo:

$$z_h = \prod_{j=0}^{r-1} w_{h,j} \text{ mód } m \qquad \forall h \in \mathbb{Z}_s$$

3. El servidor devuelve a U los $z_0, z_1, \ldots, z_{s-1} \in \mathbb{Z}_m$.

4. U solo considerará z_{i_1} porque es el correspondiente a la fila del bit en el que está interesado. Si z_{i_1} es residuo cuadrático $x_{i_1,i_2} = 0$, si no lo es $x_{i_1,i_2} = 1$. U puede conocer si es residuo cuadrático o no ya que conoce la descomposición de n en factores primos.

Proposición 4.17. *En este protocolo se cumple la propiedad de corrección.*

Demostración. La corrección se deduce de que todos los $y_0, y_1, \ldots, y_{r-1}$ son residuos cuadráticos excepto y_{i_2}. El servidor calcula la nueva matriz $\{w_{h,j}\}_{h \in \mathbb{Z}_s, j \in \mathbb{Z}_r}$. Los $w_{h,j}$, con $h \in \mathbb{Z}_s$ y $j \in \mathbb{Z}_r \backslash \{i_2\}$ serán residuos cuadráticos ya que son iguales a y_j o y_j^2 que seguirá siendo un residuo cuadrático.

Para los w_{h,i_2}, con $h \in \mathbb{Z}_s$ sólo serán residuos cuadráticos aquellos cuyo $x_{h,i_2} = 0$ porque se eleva al cuadrado el y_{i_2} correspondiente. Por ello al calcular z_h para el cual se multiplican todos los elementos de la fila $h \in \mathbb{Z}_s$, todos son residuos cuadráticos excepto el w_{h,i_2} que lo será si $x_{h,i_2} = 0$. Por lo tanto U al considerar z_{i_1} si es residuo cuadrático $x_{i_1,i_2} = 0$ y si no $x_{i_1,i_2} = 1$. Deduciéndose la corrección del esquema. □

Proposición 4.18. *En este protocolo se cumple la propiedad de privacidad.*

Demostración. Demostrémoslo por reducción al absurdo. Supongamos que el servidor puede distinguir las consultas de dos bits diferentes cuyos índices son i e i'. A partir de esta hipóstesis construiremos un circuito lógico que compute si un número es residuo cuadrático o no con el que llegaremos a contradicción.

Como la base de datos está escrita en forma de matriz $s \times r$, $i \equiv (i_1, i_2)$ e $i' \equiv (i'_1, i'_2)$, con $i_2 \neq i'_2$ porque si no el protocolo realiza las mismas operaciones por cómo está definido.

Como el servidor es capaz de distinguir las consultas sobre los índices i e i', existe una familia de circuitos de tiempo polinomial, B, tal que si B obtiene consultas sobre el índice i devuelve 1 con probabilidad p. Si las obtiene del índice i' devuelve 1 con probabilidad $p + \epsilon$, con ϵ no despreciable. Esto es $P(B(Q(i, u)) = 1) = p$ y $P(B(Q(i', u)) = 1) = p + \epsilon$, rompiendo la propiedad de privacidad 2*., como se desea.

Si U quiere recuperar el bit x_i cuyo índice es $i \equiv (i_1, i_2)$, envía al servidor $m \in H_t$ y $y_0, y_1, \ldots, y_{r-1} \in \mathbb{Z}_m$ tales que su símbolo de Jacobi es $\left(\frac{y_i}{m}\right) = 1$, $\forall i \in \mathbb{Z}_r$, dónde todos ellos son residuos cuadrátios menos y_{i_2}. Si deseara recuperar $x_{i'}$ cuyo índice es $i' \equiv (i'_1, i'_2)$ enviaría los mismos números al servidor pero escogiendo $y_{i'_2}$ no residuo cuadrático.

Construyamos ahora el circuito, C, que usando el circuito, B, pueda determinar si $y \in \mathbb{Z}_m$ es residuo cuadrático o no módulo m. Es decir, C se evaluará sobre m e y. El circuito C se va a basar en el B porque B es capaz de distinguir los índices de consultas diferentes. Por la construcción del protocolo esto significa que B es capaz de distinguir las distribuciones de las consultas cuya diferencia es la posición en la que está el no residuo cuadrático. Por lo tanto, B puede distinguir un residuo cuadrático de un no residuo cuadrático.

Para la construcción se toman $r - 2$ residuos cuadráticos módulo m y se posicionan en las r posiciones menos i_2 e i'_2. Se escoge $y \in \mathbb{Z}_m$ tal que su símbolo de Jacobi es $\left(\frac{y}{m}\right) = 1$, por lo que puede ser residuo cuadrático o no. Luego, se toma la posición i_2 ó i'_2 y se coloca en ella y, en la otra posición se escribe un residuo cuadrático módulo m. Se ejecuta B. Si la posición elegida es i'_2, la salida de C es la misma que la de B. Si la posición elegida es i_2 se invierte la salida de B. Invertir la salida de un ciruito signifca que en vez de devolver 1 devuelva 0 o viceversa.

Veamos la probabilidad de que C devuelva un 1 con la entrada m e y. Se pueden dar dos casos diferentes.

1. Si y es residuo cuadrático módulo m. En este caso los r números enviados son residuos cuadráticos. Por lo que el circuito B no va a poder distinguir las consultas. Tanto si y está posicionado en i_2 como en i'_2 la probabilidad de que B devuelva 1 con cualquiera de las consultas será q, que puede ser cualquier valor. Veamos la probabilidad de que el circuito C devuelva un 1:

   - Si y se posiciona en i'_2, C devolverá 1 con la misma probabilidad que B, es decir $P(C(m, y) = 1) = q$.
   - Si y se posiciona en i_2 invertirá la salida de B, es decir, $P(C(m, y) = 0) = q$. Luego la probabilidad de que C devuelva un 1 es $P(C(m, y) = 1) = 1 - q$

   Entonces, la probabilidad de que el circuito C devuelva un 1 en este caso es $\frac{1}{2}q + \frac{1}{2}(1 - q) = \frac{1}{2}$.

2. Si y no es un residuo cuadrático módulo m. En este caso sólo hay un no residuo cuadrático en posición i_2 ó i_2'. Las probabilidades de que el circuito lógico B devuelva un 1 en este caso ya se han estudiado y son $P(B(Q(i,u))=1)=p$ y $P(B(Q(i',u))=1)=p+\epsilon$. Veamos la probabilidad de que el circuito C devuelva un 1:

   - Si y se posiciona en i_2', C devolverá 1 con la misma probabilidad que B, es decir $P(C(m,y)=1)=p+\epsilon$.
   - Si y se posiciona en i_2 invertirá la salida de B, es decir, $P(C(m,y)=0)=p$. Luego la probabilidad de que C devuelva un 1 con el índice i es $P(C(m,y)=1)=1-p$.

   Entonces, la probabilidad de que el circuito C devuelva un 1 en este caso es $\frac{1}{2}(p+\epsilon)+\frac{1}{2}(1-p)=\frac{1}{2}+\frac{\epsilon}{2}$.

Con esto llegamos a la conclusión de que si y es residuo cuadrático el circuito C devuelve 1 con probabilidad $\frac{1}{2}$ y si no es residuo cuadrático devuelve 1 con probabilidad $\frac{1}{2}+\frac{\epsilon}{2}$. Esto contradice la suposición del problema de los residuos cuadráticos porque puede disitnguir si un número es residuo cuadrático o no con probabilidad mayor que el azar. Por lo tanto el circuito B del que partimos que distinguía consultas sobre diferentes índices no existe y la propiedad de privacidad se cumple. □

Teorema 4.19. *Para cada $c>0$ existe un esquema cPIR con complejidad de comunicación $\mathcal{O}(n^{\frac{1}{2}+c})$.*

Demostración. Analicemos la comunicación total llevada a cabo en este protocolo. U manda $m\in H_t$, $y_0,y_1,\ldots,y_{r-1}\in\mathbb{Z}_m$ al servidor. Todos ellos tienen longitud de t bits, es decir, en total U manda $t(r+1)$ bits. El servidor devuelve a U los $z_0,z_1,\ldots,z_{s-1}\in\mathbb{Z}_m$, es decir, ts bits. Luego la comunicación total es $t(r+s+1)$ bits.

En este protocolo la base de datos $x=x_0\ldots x_{n-1}$ se representa con una matriz $s\times r$, pero para la demostración del teorema se puede suponer que es $\lceil\sqrt{n}\rceil\times\lceil\sqrt{n}\rceil$. Entonces, la comunicación total es $\mathcal{O}(t\sqrt{n})$.

Además, podemos suponer que el parámetro de seguridad es $t=n^c$, para $c>0$, porque es mucho menor que la longitud de la base de datos. Obteniendo una comunicación total de $\mathcal{O}(n^{\frac{1}{2}+c})$. □

Observación 4.20. La privacidad del protocolo reside en que el servidor no puede conocer si un número es residuo cuadrático o no módulo m, porque sólo tiene capacidad para hacer operaciones en tiempo polinomial en m. Este problema como se ha visto es equivalente al de factorizar m, por lo tanto U tiene que asegurarse que el número m que elija no sea fácil de factorizar. Esto lo va a alcanzar escogiéndolo del conjunto H_t. Cuánto mayor sea t más difícil será factorizarlo y se obtendrá mayor seguridad, pero por la complejidad obtenida en el anterior protocolo a mayor t, mayor c y mayor complejidad. Entonces, hay que encontrar un equilibrio entre privacidad y que t no sea excesivamente grande. El protocolo se considera seguro cuando m tiene como mínimo 2048 bits.

Observación 4.21. En el apéndice E he desarrollado un ejemplo de este protocolo. Se ha escrito en los apéndices por falta de espacio ya que es necesario añadir la recursividad que se explica a continuación. Aunque se encuentre en los apéndices es interesante su lectura para entender correctamente el protocolo.

4.2.2. Recursividad

Supongamos que U está interesado en el bit x_i, como la base de datos está escrita en forma de matriz de dimensiones $s \times r$, $i \equiv (i_1, i_2) \in \mathbb{Z}_s \times \mathbb{Z}_r$. Con el protocolo original descrito arriba el servidor devuelve $z_0, z_1, \ldots, z_{s-1} \in \mathbb{Z}_m$ al usuario, pero U sólo necesita z_{i_1} para conocer $x_{(i_1,i_2)}$. Por lo tanto, el servidor devuelve muchos más bits de los que U necesita. Además, U no puede revelar que desea z_{i_1} porque se rompería la propiedad de privacidad. Entonces, aplicando recursivamente el protocolo cPIR se obtendrá z_{i_1} y con una complejidad de comunicación menor.

Denotaremos con P_l a cada uno de los protocolos recursivos que se ejecutan, con $l = \{L, L-1, \ldots, 1\}$, es decir, en total hay L niveles de recursividad. Para cada uno de los protocolos P_l se definen los siguientes parámetros: n_l que es el número de bits de la nueva base de datos sobre la que se ejecuta el protocolo, r_l es el número de columnas al escribir la nueva base de datos en forma de matriz es y finalmente $s_l = \frac{n_l}{r_l}$ es el número de filas. Veamos como se realiza la recursividad.

Inicialmente se ejecuta P_l con $l = L$ para el cuál $n_L = n$, es decir, se ejecuta el protocolo inicial explicado en 4.2.1, pero sólo hasta el paso 3, en ese momento el servidor no devuelve los $z_0, z_1, \ldots, z_{s-1} \in \mathbb{Z}_m$ a U. El usuario quiere recuperar z_{i_1} que por la explicación del protocolo original está formado por t bits. Para recuperarlo, ejecuta t veces el protocolo P_{L-1}. Para cada ejecucción el servidor devolvería $z_0, z_1, \ldots, z_{s_{L-1}-1} \in \mathbb{Z}_m$ con z_j para $j \in \mathbb{Z}_{s_{L-1}}$ diferentes en cada una de las veces que se ha ejecutado. Nuevamente, de cada respuesta de las t ejecuciones quiere recuperar un $z_i \in \mathbb{Z}_m$ con $i \in \mathbb{Z}_{s_{L-1}}$, lo realiza de nuevo recursivamente aplicando el protocolo P_{L-2}. Veamos como se hace esta recursión para que sea efectiva.

Partimos de que se ha ejecutado P_L para $n_L = n$, el servidor en este protocolo devolvería $z_0, z_1, \ldots, z_{s-1} \in \mathbb{Z}_m$ al usuario. Como el coste computacional es grande se recupera z_{i_1} recursivamente. Se ejecuta el siguiente paso 3* de forma recursiva para cada protocolo P_l primero con $l = L-1$, luego $l = L-2$, así hasta $l = 1$:

3*. En este paso el servidor devolvería $z_0, z_1, \ldots, z_{s_l-1} \in \mathbb{Z}_m$ a U cada uno formado por t bits y de los cuáles U sólo necesitaría un z_i con $i \in \mathbb{Z}_{s_l}$. En vez de devolverlos, se utiliza como la nueva base de datos que tiene $n_{l-1} = ts_l$ bits y se dispone en forma de matriz con r_{l-1} columnas y s_{l-1} filas. Sobre ella se aplica el protocolo PIR, P_{l-1}, t veces, denotadas como $P^1_{l-1}, P^2_{l-1}, \ldots, P^t_{l-1}$, porque se quiere recuperar z_i con $i \in Z_{s_l}$ que tiene t componentes.

Observación 4.22. En cada una de las veces que se ejecuta la recursividad del protocolo cPIR el número compuesto m es el mismo, ya que se envió al servidor en el paso 1 y se mantiene igual en todos los niveles de la recursividad. Entonces en cada una de las consultas el servidor sólo tiene que mandar los $y_0, y_1, \ldots, y_{r_l-1}$ tales que todos tienen símbolo de Jacobi 1 módulo m y son residuos cuadráticos excepto el correspondiente a la columna del bit que se desea recuperar que no será residuo cuadrático.

En la primera aplicación del protocolo P_{l-1} que es P^1_{l-1}, se obtiene el primer bit de z_i, en la segunda, P^2_{l-1}, el segundo y así hasta el último bit en P^t_{l-1}. Por este razonamiento se puede mejorar aún más este paso 3* para que en cada una de las t veces que se utiliza el protocolo P_{l-1} sea para una cadena de longitud s_l que contiene los bits en posición d de

cada z_j con $j \in \mathbb{Z}_s$ y $d \in [t]$. Esto es porque en cada ejecución P_{l-1}^d con $d \in [t]$ se obtiene el bit de dicha posición d. Lo que haremos será en cada ejecución dividir la base de datos en t columnas. Veamos esta forma más eficiente.

Cada uno de los $z_0, z_1, \ldots, z_{s_l-1} \in \mathbb{Z}_m$ está formado por t bits, por lo que los podemos representar así:

$$\begin{matrix} z_{01} & z_{02} & \cdots & z_{0t} \\ z_{11} & z_{12} & \cdots & z_{1t} \\ \vdots & \vdots & \ddots & \vdots \\ z_{s_l-11} & z_{s_l-12} & \cdots & z_{s_l-1t} \end{matrix}$$

Denotaremos con $z_{s_l}^d$ a la columna d-ésima que contiene el bit d-ésimo de cada z_j con $d \in [t]$ y $j \in \mathbb{Z}_{s_l}$. Se define el nuevo paso 3 como:

3**. Se ejecuta t veces el protocolo P_{l-1}, $P_{l-1}^1, P_{l-1}^2, \ldots, P_{l-1}^t$. Cada protocolo P_{l-1}^d con $d \in [t]$ se ejecutará para la columna $z_{s_l}^d$, es decir, para una base de datos de longitud $n_{l-1} = s_l$. De la misma manera se escribirá en forma de matriz con r_{l-1} columnas y s_{l-1} filas. Así como antes, en la ejecución número d obtiene el bit en posición d que desea de z_i con $i \in Z_{s_l}$.

Además, se desea que la base de datos utilizada en cada nivel de la recursividad sea menor, entonces, se toma $r_l = n^{\frac{1}{L+1}}$, $\forall l \in \{L, L-1, \ldots, 1\}$. Se define por recursividad el número de bits de la base de datos en el nivel l como $n_l = n^{\frac{l+1}{L+1}}$, con esta definición se cumple que $n_L = n$. Además, se consigue que al ejecutar el protocolo P_1, correspondiente al último nivel de la recursividad, que se basa en el P_0, este último se ejecute para una base de datos de $n_0 = n^{\frac{1}{L+1}}$ bits. Como el número de columnas siempre es $r_l = n^{\frac{1}{L+1}}$, entonces la base de datos se escribe en una matriz de 1 fila y $n^{\frac{1}{L+1}}$ columnas. En resumen, en el último nivel de la recursividad, P_1, se obtiene $s_0 = 1$ que es un gran avance ya que sólo se devolverán $ts_0 = t$ bits y en los niveles anteriores $s_l > 1$, para $l \in \{L, L-1, \ldots, 2\}$.

Observación 4.23. Con este nuevo paso 3** para todas las ejecuciones del protocolo P_l con $l \in \{L, L-1, \ldots, 1\}$ el bit de interés está siempre en la misma posición. Esto es consecuencia de que el usuario quiere recuperarar z_i con $i \in \mathbb{Z}_{s_l}$, que es lo mismo que recuperar $z_{i1}, z_{i2} \ldots, z_{i_t}$. En cada una de las t veces que ejecuta el protocolo P_{l-1} usando la base de datos $z_{s_l}^d$ el bit z_{id} con $d \in [t]$ siempre está en posición $i \in \mathbb{Z}_{s_l}$. Entonces, al escribir los bits en forma de matriz con r_l columnas siempre quedarán en la misma posición. Como consecuencia, para las t ejecuciones del protocolo de cada nivel $l \in \{L, L-1, \ldots, 1\}$ la consulta de U es la misma, envía $y_0, y_1, \ldots, y_{r_l-1}$ tal que $\left(\frac{y_j}{m}\right) = 1$, $\forall j \in \mathbb{Z}_{r_l}$ y todos ellos son residuos cuadráticos excepto el correspondiente a la columna del bit que se desea recuperar.

Proposición 4.24. *En este protocolo recursivo se verifican las propiedades de privacidad y corrección.*

Demostración. La corrección del protocolo recursivo se deduce de la corrección demostrada para el protocolo original en la Proposición 4.17. La diferencia es que tiene que ir descifrando desde los niveles más bajos de la recursividad hasta el primero, esto es, desde $l = 1$ hasta $l = L$.

La demostración de la privacidad se realiza de manera análoga a la del protocolo sin recurrencia, siemplemente hay que adaptarla a la nueva consulta de U. □

Teorema 4.25. *Para cada $c > 0$ existe un esquema cPIR con complejidad de comunicación $\mathcal{O}(n^c)$.*

Demostración. Veamos la comunicación total llevada a cabo entre usuario y servidor.

Primero veamos la comunicación total del usuario al servidor. Por la Observación 4.23, para cada nivel $l \in \{L, L-1, \ldots, 1\}$ el usuario manda la misma consulta para todas las veces que se tiene que ejecutar el protocolo, como sólo hay un servidor, la envía una sola vez. La consulta en cada nivel está formada por tantos números de t bits como columnas haya en dicho nivel de la recursividad, como el número de columnas siempre es $r_l = n^{\frac{1}{L+1}}$. Entonces U envia para cada nivel $n^{\frac{1}{L+1}}$ números de t bits, como en total hay L niveles en toda la recursividad U envía $t \cdot L \cdot n^{\frac{1}{L+1}}$ bits.

Ahora veamos del servidor al usuario. Hay que tener en cuenta que en la primera ejecucción del protocolo P_L sobre la base de datos inicial de longitud n, en el paso 3** se ejecuta t veces el protocolo P_{L-1}, obteniendo por cada una de ellas una nueva base de datos $n_{L-2} = s_{L-1}t$. Esta base de datos se divide entre t como se ha explicado en 3** y por cada una de ellas se ejecuta P_{L-2} obteniendo una nueva base de datos $n_{L-3} = s_{L-2}t$. Por lo tanto el protocolo P_{L-2} se ha aplicado a t cadenas distintas y por cada una de esas se obtienen t más, es decir, t^2 cadenas, como ese proceso se repite L veces finalmente se tendrán t^L cadenas. Veamos la longitud de estas cadenas. El último protocolo que se ejecuta es P_1 que se basa en la aplicación de t veces el protocolo P_0. El protocolo P_0 se ejecuta para una base de datos de longitud $n_0 = n^{\frac{1}{L+1}}$, como $r_0 = n^{\frac{1}{L+1}}$, entonces el protocolo P_0 se ejecuta t veces para una base de datos en forma de matriz $1 \times n^{\frac{1}{L+1}}$, es decir, el servidor en el último nivel de la recursividad devuelve al usuario $t^L \cdot n^{\frac{1}{L+1}}$ bits.

Sumando ambas complejidades de comunicación, la total es de $n^{\frac{1}{L+1}}(tL + t^L) \in \mathcal{O}(t^L n^{\frac{1}{L+1}})$. Si se escoge $L \in \mathcal{O}\left(\sqrt{\frac{\log_2 n}{\log_2 t}}\right)$, los términos $n^{\frac{1}{L+1}}$ y t^L son casi iguales, veámoslo sustituyendo L en t^L:

$$t^L = t^{\sqrt{\frac{\log_2 n}{\log_2 t}}} = \left(t^{\frac{\log_2 n}{\log_2 t}}\right)^{\frac{1}{\sqrt{\frac{\log_2 n}{\log_2 t}}}} \overset{1)}{=} \left(t^{\log_t n}\right)^{\frac{1}{\sqrt{\frac{\log_2 n}{\log_2 t}}}} = (n)^{\frac{1}{\sqrt{\frac{\log_2 n}{\log_2 t}}}}$$

En 1) se ha usado que $\log_t n = \frac{\log_2 n}{\log_2 t}$

El resultado obtenido cuando se sustituye $L \in \mathcal{O}\left(\sqrt{\frac{\log_2 n}{\log_2 t}}\right)$ en t^L es prácticamente igual al de $n^{\frac{1}{L+1}}$ sustituyendo L. Por lo tanto hemos conseguido una complejidad de comunicación de $\mathcal{O}\left(n^{\frac{2}{\left(\sqrt{\frac{\log_2 n}{\log_2 t}}\right)+1}}\right)$ y sustituyendo aquí $t = n^c$ con $c > 0$ se obtiene $\mathcal{O}\left(n^{\frac{2}{\left(\sqrt{\frac{\log_2 n}{c\log_2 n}}\right)+1}}\right) = \mathcal{O}\left(n^{\frac{2}{\left(\sqrt{\frac{1}{c}}\right)+1}}\right) = \mathcal{O}\left(n^{\sqrt{c}}\right)$. Renombrando $c = c^2$ se obtiene la complejidad deseada □

Observación 4.26. Una vez desglosada la recursión para calcula la complejidad de comunicación, se deduce que la recursividad se puede implementar en un protocolo de una ronda. Inicialmente U mandará $t \cdot L \cdot n^{\frac{1}{L+1}}$ bits y el servidor responderá en el último nivel de la recursividad con $\mathcal{O}(t^L n^{\frac{1}{L+1}})$ bits.

Gracias a la limitación computacional de los servidores la complejidad de comunicación obtenida es mejor que aquellas de los protocolos PIR con seguridad incondicional.

Capítulo 5

Conclusiones

En el presente trabajo se ha estudiado el método criptográfico PIR. De los diferentes tipos que existen, se ha expuesto el PIR con seguridad incondicional y el PIR computacional. El objetivo principal en el desarrollo del trabajo ha sido exponer en qué consisten estos protocolos y cómo diferentes herramientas matemáticas se emplean en el objetivo, fundamental en este ámbito, de la reducción de la complejidad de comunicación cumpliendo siempre las propiedades de privacidad y corrección.

Los primeros en estudiar el PIR con seguridad incondicional fueron B. Chor, E. Kushilevitz, O. Goldreich, y M. Sudan en 1995 [6], que consiguieron reducir la comunicación total a $\mathcal{O}\left(k' \log_2 k n^{\frac{1}{\log_2 k' + \log_2 \log_2 k}}\right)$ gracias a la utilización de códigos de recubrimiento. Posteriormente, Ambainis redujo el orden de comunicación aplicando la recursividad. Finalmente, en 2002 A. Beimel, Y. Ishai, E. Kushilevitz y J-F Raymond obtuvieron el mejor orden de comunicación para este tipo de PIR, $\mathcal{O}\left(n^{\frac{2\log_2 \log_2 k}{k log_2 k}}\right)$ para $k \geq 3$.

Respecto al PIR computacional, los primeros en introducirlo fueron E. Kushilevitz y R. Ostrovsky en 1997 [17]. Publicaron un protocolo en el que, asumiendo que el servidor no puede resolver el problema de los residuos cuadráticos y utilizando la recursividad, obtuvieron una complejidad de $\mathcal{O}(n^c)$ para $c > 0$. Una característica clave a descatar de este protocolo es que no es necesaria la replicación de la base de datos en varios servidores, se desarrolla con uno solo. También en 1997, B. Chor y N. Gilboa publicaron en [5] un protocolo para dos servidores que, utilizando funciones en un sólo sentido o funciones hash, permite obtener un orden de comunicación de $\mathcal{O}(n^\epsilon)$ para $\epsilon > 0$. Por falta de espacio no ha podido ser incluido en el trabajo.

Es importante destacar que a lo largo de todo el trabajo se han desarrollado protocolos PIR para k servidores, modelando la base de datos como una cadena de n bloques cada uno de ellos formado por 1 bit, esto lo podemos denotar $PIR_k(n, 1)$. Si tomásemos un ejemplo más realista los protocolos se aplicarían a bases de datos formadas por bloques de m bits, con $m \in \mathbb{N}$, es decir, $PIR_k(n, m)$. Para llevar a cabo un protocolo PIR en ese caso se aplicaría m veces el protocolo $PIR_k(n, 1)$. Aunque cabe mencionar que en [7] se estudió este problema para mejorar la comunicación total. Una solución es que el servidor $j \in \mathbb{Z}_k$ que en el protocolo $PIR_k(n, 1)$ devolvía $\beta_j(n)$ bits ahora en el protocolo $PIR_k(n, m)$ devuelva $m\beta_j(n)$ bits. También en el Capítulo 4 de [7] se estudia una mejora de comunicación mayor pero sólo válida para el caso en que en el $PIR_k(n, 1)$ cada servidor

devuelva 1 bit y U obtenga el bit que desea haciendo la operación XOR de todos los bits recibidos por todos los servidores.

Como se ha mencionado al principio de las conclusiones, en el trabajo se han incluido dos tipos de PIR. Existen más variantes de PIR que se podrían estudiar y que no se han incluido al ser más específicas y de uso más restringido. Algunas son:

- Protocolos PIR simétricos (SPIR) en los cuáles el usuario sólo puede obtener el bit x_i que desea y ninguno más, es decir, se garantiza también la privacidad de la base de datos, estudiados en [11] con seguridad incondicional y en [16] con seguridad computacional.
- Protocolos PIR con hasta t servidores que confabulan (véase Definición 1.2), con $t < k$. Por ejemplo en [9] se realiza almacenando la base de datos codificada.
- Protocolos PIR cuánticos (QPIR). Para entender esto es conveniente leer primero el Capítulo 3 de [1] para tener las nociones de cuántica necesarias, además en este documento se encuentra la explicación de recuperación de bits utilizando técnicas cuánticas. Para la recuperación de estados cuánticos mediante técnicas cuánticas se tiene [22].

La siguiente tabla es un resumen de los procotolos que hemos descrito y su complejidad. En ella podemos visualizar que la complejidad total de cada protocolo va disminuyendo en comparación con el anterior:

Protocolo	Comunicación total
2.1 Protocolo para k servidores	$\mathcal{O}\left(kn^{\frac{1}{\log_2 k}}\right)$
2.2 PIR basado en códigos de recubrimiento	$\mathcal{O}\left(k' \log_2 k n^{\frac{1}{\log_2 k' + \log_2 \log_2 k}}\right)$
2.3 PIR de Ambainis	$\mathcal{O}\left(2^{k^2} n^{\frac{1}{2k-1}}\right)$
3 PIR de Beimel et al.	$\mathcal{O}\left(n^{\frac{2\log_2 \log_2 k}{k log_2 k}}\right)$ para $k \geq 3$
4.2 cPIR Residuos Cuadráticos	$\mathcal{O}(n^c)$

Cuadro 5.1: Protocolos PIR y su complejidad

Apéndice A

Ejemplo PIR basado en códigos de recubrimiento

Ejemplo A.1. Ahora veamos un ejemplo del protocolo PIR basado en códigos de recubrimiento. Sea una base de datos con $n = 64$ bits.

Primero hay que elegir el número total de servidores, escogiendo el parámetro $d \in \mathbb{N}$. Hay que elegirlo estratégicamente para obtener la menor complejidad de comunicación posible. Esto lo podemos conseguir minimizando la función $\frac{2^d}{d+1} d n^{\frac{1}{d}}$ con $n = 64$. El valor que la minimiza es $d = 2,24$. Como $d = 2^m - 1$ para $m \in \mathbb{N}$, tomaremos $d = 3$ que es el valor más cercano. Por ello, el número total de servidores será $k = 2^d = 2^3 = 8$ y el de participante $k' = \frac{2^d}{d+1} = \frac{8}{4} = 2$. Consecuentemente, las tuplas de los subíndices de los bits de la base de datos tienen que ser de longitud $d = 3$. Busquemos la base en la que se tienen que escribir para conseguirlo, operando $l = \lceil \sqrt[d]{n} \rceil = \lceil \sqrt[3]{64} \rceil = 4$. Entonces, el subíndice de cada bit de la base de datos se escribe en base 4. Se presenta la base de datos de la siguiente manera:

1 1 0 1	0 0 1 1	0 0 0 0	1 1 1 1
0 1 1 1	0 0 0 0	1 0 1 0	0 1 1 0
1 0 0 1	0 0 1 0	0 0 0 1	1 0 1 1
1 1 1 1	0 1 1 0	0 1 1 0	0 1 1 0

Con esta presentación es sencillo encontrar un bit x_j cuyo subíndice esté escrito en base $l = 4$ como $(j_1, j_2, j_3) \in \mathbb{Z}_4^3$. La primera componente hace referencia a la fila, la segunda al bloque de columnas y la tercera a la posición en dicho bloque, todas ellas numeradas entre 0 y $l - 1 = 3$.

Supongamos que U quiere recuperar el bit cuyo índice es $i = 25 \equiv (1, 2, 1)_4 = (i_1, i_2, i_3)$ que es $x_{121} = 0$, marcado en azul en la base de datos.

Hay que escoger un código de recubrimiento de $\mathbb{F}_2^3$ de todos los posibles que hay. Por el Teorema 2.23 hay 4 códigos de recubrimiento posibles que son los obtenidos en el Ejemplo 2.26. Para este ejemplo escogeremos el primero que es $\{(0, 0, 0), (1, 1, 1)\}$, aunque se podría escoger cualquiera de los 4. Entonces, los servidores participantes se denotarán S_{000} y S_{111} y simularán cada uno de ellos a aquellos cuyo subíndice esté a distancia de Hamming 1 del suyo.

U escoge los subconjuntos W_j^0 con $j \in [3]$ aleatoriamente como subconjuntos de $\mathbb{Z}_4$ y define $W_j^1 = W_j^0 \oplus \{i_j\}$ con $j \in [3]$:

$$\begin{array}{ll} W_1^0 = \{1,2\} & W_1^1 = W_1^0 \oplus \{1\} = \{2\} \\ W_2^0 = \{0,2,3\} & W_2^1 = W_2^0 \oplus \{2\} = \{0,3\} \\ W_3^0 = \{0,1\} & W_3^1 = W_3^0 \oplus \{1\} = \{0\} \end{array}$$

U manda W_1^0, W_2^0 y W_3^0 a S_{000} y W_1^1, W_2^1 y W_3^1 a S_{111}. Es importante destacar que no sustituiremos los bits en el ejemplo para que se visualice perfectamente cómo se recupera el bit deseado.

El primer bit que manda S_{000} es el que corresponde a él:

$$\bigoplus_{j_1 \in W_1^0, j_2 \in W_2^0, j_3 \in W_3^0} x_{j_1,j_2,j_3} =$$

$$= x_{100} \oplus x_{101} \oplus x_{120} \oplus x_{121} \oplus x_{130} \oplus x_{131} \oplus x_{200} \oplus x_{201} \oplus x_{220} \oplus x_{221} \oplus x_{230} \oplus x_{231} \quad \text{(A.1)}$$

Ahora S_{000} simulará a todos los servidores cuyo subíndice esté en base 2 a distancia de Hamming 1 de su subíndice que es $(0,0,0) \in \mathbb{F}_2^3$. Por tanto simulará a S_{100}, S_{010} y S_{001}.

1. Veamos cómo simula al servidor S_{100}. Al hipotético S_{100} le habrían llegado W_1^1, W_2^0, W_3^0. S_{000} ha recibido el segundo y el tercero, pero no el primero, sino W_1^0. Sólo sabe que W_1^1 es de la forma $W_1^1 = W_1^0 \oplus \{i_1\}$. Como no conoce i_1, realiza secuencialmente la operación XOR $W_1^0 \oplus \{h\}$ con todos los elementos $h \in \mathbb{Z}_l$.

   Para cada $h \in \mathbb{Z}_4$, realiza la operación XOR para los bits x_{j_1,j_2,j_3}, donde $j_1 \in W_1^0 \oplus \{h\}$, $j_2 \in W_2^0$ y $j_3 \in W_3^0$, y genera las cuatro posibles respuestas del servidor simulado S_{100}.

   Sólo desarrollo explícitamente la segunda porque es la única que usará U para hacer XOR:

   $$\bigoplus_{j_1 \in \{2\}, j_2 \in \{0,2,3\}, j_3 \in \{0,1\}} x_{j_1,j_2,j_3} = x_{200} \oplus x_{201} \oplus x_{220} \oplus x_{221} \oplus x_{230} \oplus x_{231} \quad \text{(A.2)}$$

   Finalmente S_{000} envía en orden las cuatro posibles respuestas del servidor simulado S_{100}. Como el usuario, U, sabe cuál es el índice i_1, sabe con cuál de las cuatro se tiene que quedar y puede descartar el resto.

2. Análogamente, para simular S_{010} realiza la operación XOR 4 veces, para los bits x_{j_1,j_2,j_3}, donde $j_1 \in W_1^0$, $j_2 \in W_2^0 \oplus \{h\}$ y $j_3 \in W_3^0$ variando $h \in \mathbb{Z}_4$ en cada una de las cuatro respuestas.

   Sólo desarrollo explícitamente la tercera porque es la única que usará U para hacer XOR:

   $$\bigoplus_{j_1 \in \{1,2\}, j_2 \in \{0,3\}, j_3 \in \{0,1\}} x_{j_1,j_2,j_3} = x_{100} \oplus x_{101} \oplus x_{130} \oplus x_{131} \oplus x_{200} \oplus x_{201} \oplus x_{230} \oplus x_{231} \quad \text{(A.3)}$$

3. Para simular S_{001} realiza la operación XOR 4 veces, para los bits x_{j_1,j_2,j_3}, donde $j_1 \in W_1^0$, $j_2 \in W_2^0$ y $j_3 \in W_3^0 \oplus \{h\}$ variando $h \in \mathbb{Z}_4$ en cada una de las cuatro respuestas $i \in \mathbb{Z}_4$.

Sólo desarrollo explícitamente la segunda porque es la única que usará U para hacer XOR:

$$\bigoplus_{j_1\in\{1,2\},j_2\in\{0,2,3\},j_3\in\{0\}} x_{j_1,j_2,j_3} = x_{100} \oplus x_{120} \oplus x_{130} \oplus x_{200} \oplus x_{220} \oplus x_{230} \tag{A.4}$$

Veamos la respuesta de S_{111}. El primer bit que manda S_{111} es el correspondiente a todos los índices que ha recibido:

$$\bigoplus_{j_1\in\{2\},j_2\in\{0,3\},j_3\in\{0\}} x_{j_1,j_2,j_3} = x_{200} \oplus x_{230} \tag{A.5}$$

Análogamente, S_{111} simula a S_{011}, S_{101} y S_{110}.

1. Para simular a S_{011} realiza la operación XOR 4 veces, para los bits x_{j_1,j_2,j_3}, dónde $j_1 \in W_1^1 \oplus \{h\}$, $j_2 \in W_2^1$ y $j_3 \in W_3^1$ variando $h \in \mathbb{Z}_4$ en cada una de las cuatro respuestas.

   Sólo desarrollo explícitamente la segunda porque es la única que usará U para hacer XOR:

$$\bigoplus_{j_1\in\{1,2\},j_2\in\{0,3\},j_3\in\{0\}} x_{j_1,j_2,j_3} = x_{100} \oplus x_{130} \oplus x_{200} \oplus x_{230} \tag{A.6}$$

2. Para simular S_{101} realiza la operación XOR 4 veces, para los bits x_{j_1,j_2,j_3}, donde $j_1 \in W_1^1$, $j_2 \in W_2^1 \oplus \{h\}$ y $j_3 \in W_3^1$ variando $h \in \mathbb{Z}_4$ en cada una de las cuatro respuestas.

   Sólo desarrollo explícitamente la tercera porque es la única que usará U para hacer XOR:

$$\bigoplus_{j_1\in\{2\},j_2\in\{0,2,3\},j_3\in\{0\}} x_{j_1,j_2,j_3} = x_{200} \oplus x_{220} \oplus x_{230} \tag{A.7}$$

3. Para simular S_{110} realiza la operación XOR 4 veces, para los bits x_{j_1,j_2,j_3}, donde $j_1 \in W_1^1$, $j_2 \in W_2^1$ y $j_3 \in W_3^1 \oplus \{h\}$ variando $h \in \mathbb{Z}_4$ en cada una de las cuatro respuestas.

   Sólo desarrollo explícitamente la segunda porque es la única que usará U para hacer XOR:

$$\bigoplus_{j_1\in\{2\},j_2\in\{0,3\},j_3\in\{0,1\}} x_{j_1,j_2,j_3} = x_{200} \oplus x_{201} \oplus x_{230} \oplus x_{231} \tag{A.8}$$

Para recuperar el bit deseado, U hace XOR de los siguientes bits. Son los correspondientes al primero enviado por cada servidor participante y a los elegidos de las simulaciones de los servidores que están a distancia 1. Escribiremos cada respuesta que ha obtenido U en una línea diferente y referenciada, para ayudar a diferenciar los bits de cada una. Así se visualiza que todos los bits están repetidos un número par de veces

salvo x_{121} que está solo una vez. Cada bit repetido se ha cancelado con el mismo color obteniendo el bit deseado:

(A.1) $\cancel{x_{100}} \oplus x_{101} \oplus \cancel{x_{120}} \oplus x_{121} \oplus \cancel{x_{130}} \oplus \cancel{x_{131}} \oplus x_{200} \oplus \cancel{x_{201}} \oplus \cancel{x_{220}} \oplus \cancel{x_{221}} \oplus \cancel{x_{230}} \oplus \cancel{x_{231}} \oplus$

(A.2) $\oplus\, x_{200} \oplus \cancel{x_{201}} \oplus \cancel{x_{220}} \oplus \cancel{x_{221}} \oplus \cancel{x_{230}} \oplus \cancel{x_{231}} \oplus$

(A.3) $\oplus\, \cancel{x_{100}} \oplus x_{101} \oplus \cancel{x_{130}} \oplus \cancel{x_{131}} \oplus x_{200} \oplus \cancel{x_{201}} \oplus \cancel{x_{230}} \oplus \cancel{x_{231}} \oplus$

(A.4) $\oplus\, \cancel{x_{100}} \oplus \cancel{x_{120}} \oplus \cancel{x_{130}} \oplus x_{200} \oplus \cancel{x_{220}} \oplus \cancel{x_{230}} \oplus$

(A.5) $\oplus\, x_{200} \oplus \cancel{x_{230}} \oplus$

(A.6) $\oplus\, \cancel{x_{100}} \oplus \cancel{x_{130}} \oplus x_{200} \oplus \cancel{x_{230}} \oplus$

(A.7) $\oplus\, x_{200} \oplus \cancel{x_{220}} \oplus \cancel{x_{230}} \oplus$

(A.8) $\oplus\, x_{200} \oplus \cancel{x_{201}} \oplus \cancel{x_{230}} \oplus \cancel{x_{231}} =$

$= x_{121}$

Apéndice B

Ejemplo PIR Ambainis para $k = 2$ servidores

Ejemplo B.1. Veamos cómo funciona el protocolo de Ambainis para $k = 2$ servidores participantes. Supongamos que usamos una base de datos con $n = 27$ bits.

Analicemos cuál es el valor de k que minimiza la comunicación total para una base de datos de 27 bits. Para ello tenemos que minimizar la función de complejidad que es $2^{2k} n^{\frac{1}{2k-1}}$ cuando $n = 27$, el mínimo se alcanza en $k = 1,67$. Es decir, redondeando debemos escoger $k = 2$ para obtener la menor complejidad posible.

Como $k = 2$, entonces $2k - 1 = 3$, es decir, con 2 servidores participantes hay que simular a $2^{2k-1} = 8$ servidores en total, los servidores participantes serán S_{000} y S_{111}. Para el correcto funcionamiento del protocolo $l = \lceil \sqrt[3]{n} \rceil = \lceil \sqrt[3]{27} \rceil = 3$, de esta manera el subíndice de cada bit se escribe en base 3. Podemos escribir la base de datos como:

$$\begin{array}{ccc} 1\,0\,1 & 1\,1\,0 & 1\,0\,1 \\ 0\,1\,0 & 1\,0\,0 & 0\,1\,1 \\ 1\,0\,0 & 1\,0\,0 & 0\,1\,1 \end{array}$$

Con esta escritura es sencillo encontrar un bit x_j cuyo subíndice esté escrito en base 3 como $(j_1, j_2, j_3) \in \mathbb{Z}_3^3$. La primera componente hace referencia a la fila, la segunda al bloque de columnas y la tercera a la posición en dicho bloque, todas ellas numeradas entre 0 y $l - 1 = 2$

Supongamos que U quiere recuperar el bit cuyo índice es $i = 6 \equiv (0, 2, 0)_3 = (i_1, i_2, i_3)$ que es $x_{020} = 1$, marcado en azul en la base de datos.

U escoge los subconjuntos W_j^0 con $j \in [3]$ aleatoriamente como subconjuntos de $\mathbb{Z}_3$ y define $W_j^1 = W_j^0 \oplus \{i_j\}$ con $j \in [3]$:

$$\begin{array}{ll} W_1^0 = \{0, 1\} & W_1^1 = \{0, 1\} \oplus \{0\} = \{1\} \\ W_2^0 = \{1, 2\} & W_2^1 = \{1, 2\} \oplus \{2\} = \{1\} \\ W_3^0 = \{1\} & W_3^1 = \{1\} \oplus \{0\} = \{0, 1\} \end{array}$$

U envía W_j^0 al primer servidor $\forall j \in [3]$ y W_j^1 al segundo servidor $\forall j \in [3]$.

El primer bit que manda S_{000} es el que corresponde a él:

$$\bigoplus_{j_1\in\{0,1\},j_2\in\{1,2\},j_3\in\{1\}} x_{j_1,j_2,j_3} = x_{011} \oplus x_{021} \oplus x_{111} \oplus x_{121} \tag{B.1}$$

Luego, S_{000} simulará a todos los servidores cuyo subíndice en base 2 esté a distancia de Hamming 1 de su subíndice que es $(0,0,0) \in \mathbb{F}_2^3$. Por lo tanto simulará a los servidores S_{100}, S_{010} y S_{001}. Veamos cómo simula a cada uno de ellos.

1. Primero simula al servidor S_{100}. Para ello fija $W_2^0 = \{1,2\}$ y $W_3^0 = \{1\}$ y realiza la operación XOR para los bits x_{j_1,j_2,j_3}, dónde $j_1 \in W_1^0 \oplus \{j\}$, $j_2 \in W_2^0$ y $j_3 \in W_3^0$ variando $j \in \mathbb{Z}_3$ en cada una de las 3 respuestas.

   Sólo desarrollo explícitamente la primera ya que es la que escogerá U para hacer XOR, porque es la que simula a S_{100}.

   $$\bigoplus_{j_1\in\{1\},j_2\in\{1,2\},j_3\in\{1\}} x_{j_1,j_2,j_3} = x_{111} \oplus x_{121} \tag{B.2}$$

   Finalmente S_{000} envía en orden las tres posibles respuestas del servidor simulado S_{100}. Como el usuario, U, sabe cuál es el índice i_1, sabe con cuál de las tres se tiene que quedar y puede descartar el resto.

2. Luego simula al servidor S_{010}. Para ello fija $W_1^0 = \{0,1\}$ y $W_3^0 = \{1\}$ y realiza la operación XOR para los bits x_{j_1,j_2,j_3}, dónde $j_1 \in W_1^0$, $j_2 \in W_2^0 \oplus \{j\}$ y $j_3 \in W_3^0$ variando $j \in \mathbb{Z}_3$ en cada una de las 3 respuestas.

   Sólo desarrollo explícitamente la tercera ya que es la que escogerá U para hacer XOR, porque es la que simula a S_{010}.

   $$\bigoplus_{j_1\in\{0,1\},j_2\in\{1\},j_3\in\{1\}} x_{j_1,j_2,j_3} = x_{011} \oplus x_{111} \tag{B.3}$$

3. Finalmente simula al servidor S_{001}. Para ello fija $W_1^0 = \{0,1\}$ y $W_2^0 = \{1,2\}$ y realiza la operación XOR para los bits x_{j_1,j_2,j_3}, dónde $j_1 \in W_1^0$, $j_2 \in W_2^0$ y $j_3 \in W_3^0 \oplus \{j\}$ variando $j \in \mathbb{Z}_3$ en cada una de las 3 respuestas.

   Sólo desarrollo explícitamente la primera ya que es la que escogerá U para hacer XOR, porque es la que simula a S_{001}.

   $$\bigoplus_{j_1\in\{0,1\},j_2\in\{1,2\},j_3\in\{0,1\}} x_{j_1,j_2,j_3} = x_{010}\oplus x_{011}\oplus x_{020}\oplus x_{021}\oplus x_{110}\oplus x_{111}\oplus x_{120}\oplus x_{121} \tag{B.4}$$

El servidor S_{111} inicialmente realiza la operación XOR de los siguientes bits x_{j_1,j_2,j_3}.

$$\bigoplus_{j_1\in\{1\},j_2\in\{1\},j_3\in\{0,1\}} x_{j_1,j_2,j_3} = x_{110} \oplus x_{111} \tag{B.5}$$

Luego S_{111} simulará a todos los servidores cuyo subíndice en base 2 esté a distancia de Hamming menor o igual que $2k - 3 = 1$ de su subíndice que es $(1,1,1) \in \mathbb{F}_2^3$. Por lo tanto simulará a los servidores S_{011}, S_{101} y S_{110}. Veamos cómo simula a cada uno de ellos.

1. Primero simula al servidor S_{011}. Para ello fija $W_2^1 = \{1\}$ y $W_3^1 = \{0, 1\}$ y realiza la operación XOR para los bits x_{j_1,j_2,j_3}, dónde $j_1 \in W_1^1 \oplus \{j\}$, $j_2 \in W_2^1$ y $j_3 \in W_3^1$ variando $j \in \mathbb{Z}_3$ en cada una de las 3 respuestas.

   Sólo desarrollo explícitamente la primera ya que es la que escogerá U para hacer XOR, porque es la que simula a S_{011}.

$$\bigoplus_{j_1\in\{0,1\},j_2\in\{1\},j_3\in\{0,1\}} x_{j_1,j_2,j_3} = x_{010} \oplus x_{011} \oplus x_{110} \oplus x_{111} \tag{B.6}$$

2. Luego simula al servidor S_{101}. Para ello fija fija $W_1^1 = \{1\}$ y $W_3^1 = \{0, 1\}$ y realiza la operación XOR para los bits x_{j_1,j_2,j_3}, dónde $j_1 \in W_1^1$, $j_2 \in W_2^1 \oplus \{j\}$ y $j_3 \in W_3^1$ variando $j \in \mathbb{Z}_3$ en cada una de las 3 respuestas.

   Sólo desarrollo explícitamente la tercera ya que es la que escogerá U para hacer XOR, porque es la que simula a S_{101}.

$$\bigoplus_{j_1\in\{1\},j_2\in\{1,2\},j_3\in\{0,1\}} x_{j_1,j_2,j_3} = x_{110} \oplus x_{111} \oplus x_{120} \oplus x_{121} \tag{B.7}$$

3. Finalmente simula al servidor S_{110}. Para ello fija $W_1^1 = \{1\}$ y $W_2^1 = \{1\}$ y realiza la operación XOR para los bits x_{j_1,j_2,j_3}, dónde $j_1 \in W_1^1$, $j_2 \in W_2^1$ y $j_3 \in W_3^1 \oplus \{j\}$ variando $j \in \mathbb{Z}_3$ en cada una de las 3 respuestas.

   Sólo desarrollo explícitamente la primera ya que es la que escogerá U para hacer XOR, porque es la que simula a S_{110}.

$$\bigoplus_{j_1\in\{1\},j_2\in\{1\},j_3\in\{1\}} x_{j_1,j_2,j_3} = x_{111} \tag{B.8}$$

Para obtener el bit deseado, U hace la operación XOR para todos los bits recogidos. Escribiremos cada respuesta que ha obtenido U en una línea diferente y referenciada, para ayudar a diferenciar los bits de cada una. Así se visualiza que todos los bits están repetidos un número par de veces salvo x_{020} que está solo una vez,

(B.1) $\cancel{x_{011}} \oplus \cancel{x_{021}} \oplus \cancel{x_{111}} \oplus \cancel{x_{121}} \oplus$

(B.2) $\oplus\, \cancel{x_{111}} \oplus \cancel{x_{121}} \oplus$

(B.3) $\oplus\, \cancel{x_{011}} \oplus \cancel{x_{111}} \oplus$

(B.4) $\oplus\, \cancel{x_{010}} \oplus \cancel{x_{011}} \oplus x_{020} \oplus \cancel{x_{021}} \oplus \cancel{x_{110}} \oplus \cancel{x_{111}} \oplus \cancel{x_{120}} \oplus \cancel{x_{121}} \oplus$

(B.5) $\oplus\, \cancel{x_{110}} \oplus \cancel{x_{111}} \oplus$

(B.6) $\oplus\, \cancel{x_{010}} \oplus \cancel{x_{011}} \oplus \cancel{x_{110}} \oplus \cancel{x_{111}} \oplus$

(B.7) $\oplus\, \cancel{x_{110}} \oplus \cancel{x_{111}} \oplus \cancel{x_{120}} \oplus \cancel{x_{121}} \oplus$

(B.8) $\oplus\, \cancel{x_{111}} =$

$= x_{020}$

U recupera correctamente el bit que le interesaba.

Apéndice C

Ejemplo PIR Beimel para $d = k - 1$

Ejemplo C.1. Veamos un ejemplo para $d = k - 1$. Supongamos que tenemos la base de datos $x = \{0, 1, 1\}$, es decir, $n = 3$.

Tenemos que escoger el número k de servidores para utilizar el protocolo. Para ello minimizamos la función de la complejidad de comunicación total $k^2 n^{\frac{1}{k-1}}$, el mínimo se obtiene en $k = 2,07$, pero lo tenemos que redondear a un número natural. Tomaremos $k = 2$. Por lo tanto $d = k - 1 = 1$. Ahora elegimos el valor de m de manera que $\binom{m}{d} \geq n$, por ejemplo $m = 4$. Supongamos que U está interesado en el bit x_2.

Primero U escribe aleatoriamente $E_i \in \mathbb{F}_2^4$, $\forall i \in [3]$ con peso de Hamming 1.

$$E_1 = \{1, 0, 0, 0\}$$
$$E_2 = \{0, 1, 0, 0\}$$
$$E_3 = \{0, 0, 1, 0\}$$

Entonces P_x sería: $P_x(Z_1, Z_2, Z_3, Z_4) = x_1 Z_1 + x_2 Z_2 + x_3 Z_3$ y sustituyendo $Z_h = \sum_{j=1}^{2} Y_{j,h}$ para cada $h \in [4]$ se obtiene Q_x.

$$Q_x\left(\begin{pmatrix} Y_{11} & Y_{12} & Y_{13} & Y_{14} \\ Y_{21} & Y_{22} & Y_{23} & Y_{24} \end{pmatrix}\right) = x_1(Y_{11} + Y_{21}) + x_2(Y_{12} + Y_{22}) + x_3(Y_{13} + Y_{23}).$$

En total Q_x tiene 6 monomios. Antes de empezar el protocolo se hace la asignación a los $k = 2$ servidores.

A S_1 se le asignan los monomios $x_1 Y_{21}$, $x_2 Y_{22}$ y $x_3 Y_{23}$ porque ninguna de sus variables están en Y_1.

A S_2 se le asignan los monomios $x_1 Y_{11}$, $x_2 Y_{12}$ y $x_3 Y_{13}$ porque ninguna de sus variables están en Y_2.

Después de la asignación U elige $y_1 = \{0, 1, 0, 1\}$ y construye y_2 tal que $y_1 + y_2 = E_2$. Por lo que $y_2 = \{0, 0, 0, 1\}$. U manda y_1 a S_2 y y_2 a S_1.

Veamos las respuestas de ambos servidores, en ellas no sustituyo el valor de los x_1, x_2, x_3 para que se vea perfectamente como se obtiene el bit deseado, en la práctica sí estarían sustituidos. S_1 responde con la suma de todos los monomios asignados a él, evaluándolos con las variables que conoce que, como se ha visto por ser $d = k - 1$, son todas. Devuelve $x_1 \cdot 0 + x_2 \cdot 0 + x_3 \cdot 0 = 0$. S_2 realiza el mismo cálculo con los monomios que le han sido asignados, por lo que devuelve $x_1 \cdot 0 + x_2 \cdot 1 + x_3 \cdot 0 = x_2$.

Finalmente, U suma todos los bits recibidos y no tiene que evaluar en ningún punto ya que los servidores conocían todas las variables. Así consigue el bit querido, x_2.

Apéndice D

Ejemplo PIR Beimel para $d = 2k - 1$

Ejemplo D.1. Veamos un ejemplo para $d = 2k - 1$. Desarrollamos también un ejemplo para $d = k - 1$ en el Apéndice C. Sea la base de datos $x = \{0110011\}$, es decir, $n = 7$.

Tenemos que escoger el número k de servidores para utilizar el protocolo. Para ello minimizamos la función de la complejidad de comunicación total $k^2 n^{\frac{1}{2k-1}}$, el mínimo se obtiene en $k = 1,29$, pero lo tenemos que redondear a un número natural. Tomaremos $k = 2$ porque con $k = 1$ el protocolo no será representativo. Por lo tanto $d = 2k - 1 = 3$. Ahora elegimos el valor de m de manera que $\binom{m}{d} \geq n$, por ejemplo $m = 5$. Supongamos que U está interesado en el bit x_3.

Primero U escribe aleatoriamente $E_i \in \mathbb{F}_2^5$, $\forall i \in [7]$ con peso de Hamming $d = 3$.

$$
\begin{aligned}
E_1 &= \{0,1,1,1,0\} \\
E_2 &= \{1,1,0,1,0\} \\
E_3 &= \{0,1,1,0,1\} \\
E_4 &= \{1,0,1,1,0\} \\
E_5 &= \{0,0,1,1,1\} \\
E_6 &= \{1,0,0,1,1\} \\
E_7 &= \{0,1,0,1,1\}
\end{aligned}
$$

Entonces P_x sería: $P_x(Z_1, Z_2, Z_3, Z_4, Z_5) = x_1 Z_2 Z_3 Z_4 + x_2 Z_1 Z_2 Z_4 + x_3 Z_2 Z_3 Z_5 + x_4 Z_1 Z_3 Z_4 + x_5 Z_3 Z_4 Z_5 + x_6 Z_1 Z_4 Z_5 + x_7 Z_2 Z_4 Z_5$ y sustituyendo $Z_h = \sum_{j=1}^{2} Y_{j,h}$ para cada $h \in [5]$ se obtiene Q_x.

$$
\begin{aligned}
Q_x\left(\begin{pmatrix} Y_{11} & Y_{12} & Y_{13} & Y_{14} & Y_{15} \\ Y_{21} & Y_{22} & Y_{23} & Y_{24} & Y_{25} \end{pmatrix}\right) = {} & x_1(Y_{12}+Y_{22})(Y_{13}+Y_{23})(Y_{14}+Y_{24}) + \\
& + x_2(Y_{11}+Y_{21})(Y_{12}+Y_{22})(Y_{14}+Y_{24}) + \\
& + x_3(Y_{12}+Y_{22})(Y_{13}+Y_{23})(Y_{15}+Y_{25}) + \\
& + x_4(Y_{11}+Y_{21})(Y_{13}+Y_{23})(Y_{14}+Y_{24}) + \\
& + x_5(Y_{13}+Y_{23})(Y_{14}+Y_{24})(Y_{15}+Y_{25}) + \\
& + x_6(Y_{11}+Y_{21})(Y_{14}+Y_{24})(Y_{15}+Y_{25}) + \\
& + x_7(Y_{12}+Y_{22})(Y_{14}+Y_{24})(Y_{15}+Y_{25})
\end{aligned}
$$

En total, Q_x tiene 56 monomios. Veamos como se asignan a los dos servidores.

Los monomios que tenemos que asignar pueden ser de 8 tipos. $Y_{1\star}Y_{1\star}Y_{1\star}$, $Y_{2\star}Y_{1\star}Y_{1\star}$, $Y_{1\star}Y_{2\star}Y_{1\star}$, $Y_{1\star}Y_{1\star}Y_{2\star}$, $Y_{1\star}Y_{2\star}Y_{2\star}$, $Y_{2\star}Y_{1\star}Y_{2\star}$, $Y_{2\star}Y_{2\star}Y_{1\star}$ y $Y_{2\star}Y_{2\star}Y_{2\star}$. En el segundo subíndice se ha escrito $\star$ ya que no es relevante para la asignación.

Los monomios del tipo $Y_{1\star}Y_{1\star}Y_{1\star}$ se asignan al servidor 2, ya que ninguna variable suya está en Y_2. Y los del tipo $Y_{2\star}Y_{2\star}Y_{2\star}$ al servidor 1, ya que ninguna variable está en Y_1. Ambos se rigen por la primera condición de asignación en el caso de $d = 2k - 1$.

Los demás se van a asociar a la segunda condición ya que tienen variables de Y_1 y de Y_2. Los monomios del tipo $Y_{2\star}Y_{1\star}Y_{1\star}$, $Y_{1\star}Y_{2\star}Y_{1\star}$ y $Y_{1\star}Y_{1\star}Y_{2\star}$ como solo hay una variable de Y_2 se le asigna al segundo servidor. Y los de tipo $Y_{1\star}Y_{2\star}Y_{2\star}$, $Y_{2\star}Y_{1\star}Y_{2\star}$ y $Y_{2\star}Y_{2\star}Y_{1\star}$ se le asignan al servidor 1 ya que solo hay una variable de Y_1.

Se divide la asignación de los monomios según el coeficiente que tienen en la siguiente tabla:

x_i	S_1	S_2
x_1	$Y_{12}Y_{23}Y_{24}$, $Y_{22}Y_{13}Y_{24}$, $Y_{22}Y_{23}Y_{14}$, $Y_{22}Y_{23}Y_{24}$	$Y_{12}Y_{13}Y_{14}$, $Y_{12}Y_{13}Y_{24}$, $Y_{12}Y_{23}Y_{14}$, $Y_{22}Y_{13}Y_{14}$
x_2	$Y_{11}Y_{22}Y_{24}$, $Y_{21}Y_{12}Y_{24}$, $Y_{21}Y_{22}Y_{14}$, $Y_{21}Y_{22}Y_{24}$	$Y_{11}Y_{12}Y_{14}$, $Y_{11}Y_{12}Y_{24}$, $Y_{11}Y_{22}Y_{14}$, $Y_{21}Y_{12}Y_{14}$
x_3	$Y_{12}Y_{23}Y_{25}$, $Y_{22}Y_{13}Y_{25}$, $Y_{22}Y_{23}Y_{15}$, $Y_{22}Y_{23}Y_{25}$	$Y_{12}Y_{13}Y_{15}$, $Y_{12}Y_{13}Y_{25}$, $Y_{12}Y_{23}Y_{15}$, $Y_{22}Y_{13}Y_{15}$
x_4	$Y_{11}Y_{23}Y_{24}$, $Y_{21}Y_{13}Y_{24}$, $Y_{21}Y_{23}Y_{14}$, $Y_{21}Y_{23}Y_{24}$	$Y_{11}Y_{13}Y_{14}$, $Y_{11}Y_{13}Y_{24}$, $Y_{11}Y_{23}Y_{14}$, $Y_{21}Y_{13}Y_{14}$
x_5	$Y_{13}Y_{24}Y_{25}$, $Y_{23}Y_{14}Y_{25}$, $Y_{23}Y_{24}Y_{15}$, $Y_{23}Y_{24}Y_{25}$	$Y_{13}Y_{14}Y_{15}$, $Y_{13}Y_{14}Y_{25}$, $Y_{13}Y_{24}Y_{15}$, $Y_{23}Y_{14}Y_{15}$
x_6	$Y_{11}Y_{24}Y_{25}$, $Y_{21}Y_{14}Y_{25}$, $Y_{21}Y_{24}Y_{15}$, $Y_{21}Y_{24}Y_{25}$	$Y_{11}Y_{14}Y_{15}$, $Y_{11}Y_{14}Y_{25}$, $Y_{11}Y_{24}Y_{15}$, $Y_{21}Y_{14}Y_{15}$
x_7	$Y_{12}Y_{24}Y_{25}$, $Y_{22}Y_{14}Y_{25}$, $Y_{22}Y_{24}Y_{15}$, $Y_{22}Y_{24}Y_{25}$	$Y_{12}Y_{14}Y_{15}$, $Y_{12}Y_{14}Y_{25}$, $Y_{12}Y_{24}Y_{15}$, $Y_{22}Y_{14}Y_{15}$

Luego, U elige $y_1 = \{0, 1, 0, 1, 0\}$ y construye y_2 tal que $y_1 + y_2 = E_3$. Por lo tanto, $y_2 = \{0, 0, 1, 1, 1\}$. Manda y_1 al segundo servidor e y_2 al primero.

Veamos las respuestas de ambos servidores, en ellas no sustituyo el valor de los $x_1, x_2, \ldots, x_7$ para que se vea perfectamente como se obtiene el bit deseado, en la práctica sí estarían sustituidos. El servidor S_1 contesta con la suma de todos los monomios que le han sido asignados, es decir, toda la columna correspondiente a S_1, evaluándolos en las variables que conoce. Por lo que devuelve $P_1(Y_{11}, Y_{12}, Y_{13}, Y_{14}, Y_{15}) = x_1Y_{12} + x_3Y_{12} + x_4Y_{11} + x_5(Y_{13} + Y_{14} + Y_{15} + 1) + x_6Y_{11} + x_7Y_{12}$. Lo mismo realiza el servidor S_2 devolviendo $P_2(Y_{21}, Y_{22}, Y_{23}, Y_{24}, Y_{25}) = x_1Y_{23} + x_2Y_{21} + x_7Y_{25}$.

U conoce y_1 e y_2 así que evalúa las variables que faltan. Quedando la respuesta del primer servidor $P_1(y_{11}, y_{12}, y_{13}, y_{14}, y_{15}) = x_1 + x_3 + x_7$ y la del segundo $P_2(y_{21}, y_{22}, y_{23}, y_{24}, y_{25}) = x_1 + x_7$. U suma ambas respuestas sobre $\mathbb{F}_2$ y obtiene $P_1(y_{11}, y_{12}, y_{13}, y_{14}, y_{15}) + P_2(y_{21}, y_{22}, y_{23}, y_{24}, y_{25}) = x_3$ que es el bit que quería recuperar.

Apéndice E

Ejemplo cPIR basado en residuos cuadráticos

Ejemplo E.1. Supongamos que el protocolo se va a realizar sobre una base de datos de $n = 30$ bits. Se va a escribir como una matriz $\lceil\sqrt{n}\rceil \times \lceil\sqrt{n}\rceil$. Como $\lceil\sqrt{30}\rceil = 6$ se rellenarán los 6 bits que faltan con 0 y 1 sin importar el orden o la cantidad de cada uno. Se presenta la base de datos de la siguiente manera:

$$\begin{array}{cccccc} 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 & 0 \end{array}$$

Con esta presentación es sencillo encontrar un bit x_j cuyo subíndice está esrito en base $l = 6$ como $(j_1, j_2) \in \mathbb{Z}_l^2$, la primera componente hace referencia a la fila de la matriz y la segunda a la columna. Teniendo en cuenta que la numeración es en $\mathbb{Z}_l$.

Supongamos que U quiere conocer el bit cuyo índice es $i = 15 \equiv (2, 3)_6$ que es $x_{23} = 1$, cuyos dígitos de fila y columna, son respectivamente $2, 3 \in \mathbb{Z}_l$, marcado en azul en la base de datos.

Inicialmente U escoge $m \in H_t$, para el ejemplo tomaremos $\frac{t}{2} = 7$, aunque se podría elegir mayor para obtener más seguridad siempre teniendo en cuenta que a mayor t también aumenta la comunicación total del protocolo.

Podemos tomar los números primos en binario 1000011 y 1110001 que son 67 y 113 respectivamente en decimal. Por lo tanto, $m = 1110110010011$ que es 7571 en decimal. Como la base de datos escrita en forma de matriz tiene 6 columnas, hay que elegir $y_0, y_1, \ldots, y_5$ tales que el símbolo de Jacobi $\left(\frac{y_j}{m}\right) = 1$, $\forall j \in \mathbb{Z}_6$. Todos ellos deben ser residuos cuadráticos módulo 7571 menos y_3 porque es el correspondiente a la segunda componente del índice del bit que desea. Entonces U escoge los siguientes $y_0, y_1, \ldots, y_5$ aunque podrían ser otros cualquiera, pero cumpliendo las propiedades mencionadas:

$$y_0 = 1905 \equiv (0011101110001)_2 \quad y_1 = 3245 \equiv (110010101101)_2$$
$$y_2 = 7179 \equiv (1110000001011)_2 \quad y_3 = 2975 \equiv (101110011111)_2$$
$$y_4 = 6889 \equiv (1101011101001)_2 \quad y_5 = 1691 \equiv (011010011011)_2$$

El usuario envía al servidor $m, y_0, y_1, y_2, y_3, y_4, y_5$ todos ellos en binario.

El servidor primero calcula $\{w_{h,j}\}_{h\in\mathbb{Z}_s, j\in\mathbb{Z}_r}$ como se explicó en el paso 2. Se escriben todos los elementos en decimal para que se vea cómo se calculan, aunque el servidor la hubiese calculado en binario.

$$\begin{array}{cccccc}
(1905)^2 & 3245 & 7179 & (2975)^2 & 6889 & (1691)^2 \\
1905 & (3245)^2 & 7179 & (2975)^2 & (6889)^2 & 1691 \\
(1905)^2 & 3245 & 7179 & 2975 & (6889)^2 & (1691)^2 \\
1905 & 3245 & (7179)^2 & (2975)^2 & (6889)^2 & 1691 \\
(1905)^2 & (3245)^2 & 7179 & 2975 & 6889 & 1691 \\
1905 & 3245 & 7179 & (2975)^2 & 6889 & (1691)^2
\end{array}$$

De todos los elementos se hace módulo $m = 7571$, obteniendo $\{w_{h,j}\}_{6\times 6}$.

$$\begin{array}{cccccc}
2516 & 3245 & 7179 & 126 & 6889 & 5214 \\
1905 & 6335 & 7179 & 126 & 3293 & 1691 \\
2516 & 3245 & 7179 & 2975 & 3293 & 5214 \\
1905 & 3245 & 2244 & 126 & 3293 & 1691 \\
2516 & 6335 & 7179 & 2975 & 6889 & 1691 \\
1905 & 3245 & 7179 & 126 & 6889 & 5214
\end{array}$$

Posteriormente, el servidor calcula $z_0, z_1, \ldots, z_5$ como se explicó en el paso 2, se deja el resultado en decimal y en binario para que sea más fácil su comprensión, pero el servidor lo calcularía en binario.

$$z_0 = \prod_{j=0}^{5} w_{0,j} = 1143 \equiv (10001110111)_2 \qquad z_1 = \prod_{j=0}^{5} w_{1,j} = 7032 \equiv (1101101111000)_2$$
$$z_2 = \prod_{j=0}^{5} w_{2,j} = 7197 \equiv (1110000011101)_2 \quad z_3 = \prod_{j=0}^{5} w_{3,j} = 5121 \equiv (1010000000001)_2$$
$$z_4 = \prod_{j=0}^{5} w_{4,j} = 522 \equiv (1000001010)_2 \qquad z_5 = \prod_{j=0}^{5} w_{5,j} = 3029 \equiv (101111010101)_2$$

El servidor devuelve al usuario $z_0, z_1, \ldots, z_5$, pero U sólo recoge z_2 porque es el correspondiente a la fila del índice del bit que quiere recuperar, x_{23}.

Por último, U comprueba si z_2 es residuo cuadrático o no, si lo es $x_{23} = 0$ si no $x_{23} = 1$. Vamos a comprobarlo, como U conoce la descomposición de $m = 7571 = 113 \cdot 67$, puede calcular los símbolos de Legendre de 7197 módulo 113 y módulo 67. Encima de cada igualdad se referencia la propiedad que se utiliza

$\left(\frac{7197}{113}\right) \overset{2.}{=} \left(\frac{3}{113}\right)\left(\frac{2399}{113}\right)$

Se calcula cada símbolo de Lengendre por separado.

$\left(\frac{3}{113}\right) \overset{4.}{=} \left(\frac{113}{3}\right)(-1)^{\frac{112\cdot 2}{4}} \overset{1.}{=} \left(\frac{2}{3}\right) \overset{4.}{=} (-1)^{\frac{3^2-1}{8}} = -1$

$\left(\frac{2399}{113}\right) \overset{1.}{=} \left(\frac{26}{113}\right) \overset{2.}{=} \left(\frac{2}{113}\right)\left(\frac{13}{113}\right) \overset{3.4.}{=} (-1)^{\frac{113^2-1}{8}}\left(\frac{113}{13}\right) \overset{1.}{=} \left(\frac{9}{13}\right) \overset{2.}{=} \left(\frac{3}{13}\right)^2 = 1$

Por lo tanto $\left(\frac{7197}{113}\right) = \left(\frac{3}{113}\right)\left(\frac{2399}{113}\right) = -1$, como 7197 no es residuo cuadrático módulo 113 tampoco lo va a ser módulo 7571, no es necesario calcular $\left(\frac{7197}{67}\right)$ entonces $x_{23} = 1$.

Índice de cuadros

Referencias

[1] Matteo Allaix, *Quantum private information retrieval from coded storage systems*, arXiv preprint arXiv:2312.07570 (2023).

[2] Andris Ambainis, *Upper bound on the communication complexity of private information retrieval*, International Colloquium on Automata, Languages, and Programming, Springer, 1997, pp. 401–407.

[3] Elaine Barker, *Ar transitioning the use of cryptographic algorithms and key lengths*, National Institute of Standards and Technology: Washington, DC, USA (2019).

[4] Amos Beimel, Yuval Ishai, Eyal Kushilevitz, and J-F Raymond, *Breaking the $\mathcal{O}(n^{\frac{1}{2k-1}})$ barrier for information-theoretic private information retrieval*, Proceedings 43rd Annual IEEE Symposium on Foundations of Computer Science, IEEE, 2002, pp. 261–270.

[5] Benny Chor and Niv Gilboa, *Computationally private information retrieval*, Proceedings of the twenty-ninth annual ACM symposium on Theory of computing, 1997, pp. 304–313.

[6] Benny Chor, Eyal Kushilevitz, Oded Goldreich, and Madhu Sudan, *Private information retrieval*, Proceedings of the 36th Annual IEEE Symposium on Foundations of Computer Science, IEEE, 1995, pp. 41–50.

[7] ———, *Private information retrieval*, Journal of the ACM (JACM) **45** (1998), no. 6, 965–981.

[8] Shawna Meyer Eikenberry and Jonathan P Sorenson, *Efficient algorithms for computing the jacobi symbol*, Journal of Symbolic Computation **26** (1998), no. 4, 509–523.

[9] Ragnar Freij-Hollanti, Oliver W Gnilke, Camilla Hollanti, Anna-Lena Horlemann-Trautmann, David Karpuk, and Ivo Kubjas, *t-private information retrieval schemes using transitive codes*, IEEE Transactions on Information Theory **65** (2018), no. 4, 2107–2118.

[10] William Gasarch, *A survey on private information retrieval*, Bulletin of the EATCS **82** (2004), no. 72-107, 113.

[11] Yael Gertner, Yuval Ishai, Eyal Kushilevitz, and Tal Malkin, *Protecting data privacy in private information retrieval schemes*, Proceedings of the thirtieth annual ACM symposium on Theory of computing, 1998, pp. 151–160.

[12] Shafi Goldwasser and Silvio Micali, *Probabilistic encryption & how to play mental poker keeping secret all partial information*, Providing sound foundations for cryptography: on the work of Shafi Goldwasser and Silvio Micali, 1984, pp. 173–201.

[13] Paul Halmos and Steven Givant, *Introduction to boolean algebras*, Springer, 2009.

[14] Yuval Ishai and Eyal Kushilevitz, *Improved upper bounds on information-theoretic private information retrieval*, Proceedings of the thirty-first annual ACM symposium on Theory of computing, 1999, pp. 79–88.

[15] Toshiya Itoh, *On lower bounds for the communication complexity of private information retrieval*, IEICE transactions on fundamentals of electronics, communications and computer sciences **84** (2001), no. 1, 157–164.

[16] Sanjeev Kumar Mishra and Palash Sarkar, *Symmetrically private information retrieval*, Progress in Cryptology—INDOCRYPT 2000: First International Conference in Cryptology in India Calcutta, India, December 10–13, 2000 Proceedings 1, Springer, 2000, pp. 225–236.

[17] Eyal Kushilevitz and Rafail Ostrovsky, *Replication is not needed: Single database, computationally-private information retrieval*, Proceedings 38th annual symposium on foundations of computer science, IEEE, 1997, pp. 364–373.

[18] Julio P Lafuente and Gustavo Ochoa, *Teoría de la información: códigos y criptografía*, (2021).

[19] Franz Lemmermeyer, *Hermite's identity and the quadratic reciprocity law*, Elemente der Mathematik **78** (2022), no. 4, 171–173.

[20] Francisco José Plaza Martín, *Manual de criptografía: fundamentos matemáticos de la criptografía para un estudiante de grado*, Manual de criptografía (2021), 116.

[21] Victor Shoup, *A computational introduction to number theory and algebra*, Cambridge university press, 2009.

[22] Seunghoan Song and Masahito Hayashi, *Quantum private information retrieval for quantum messages*, 2021 IEEE International Symposium on Information Theory (ISIT), IEEE, 2021, pp. 1052–1057.

[23] Sajani Vithana, Zhusheng Wang, and Sennur Ulukus, *Private information retrieval and its applications: An introduction, open problems, future directions*, arXiv preprint arXiv:2304.14397 (2023).