Sección de Matemáticas
Universidad de La Laguna

Pedro Gonzalo Méndez Hernández

Introducción a la Criptografía Homomórfica

Introduction to Homomorphic Cryptography

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

Dirigido por
Pino Teresa Caballero Gil
María Candelaria Hernández Goya

Pino Teresa Caballero Gil
Departamento de Ingeniería Informática y de Sistemas
Universidad de La Laguna
38200 San Cristóbal de La Laguna, Tenerife
España

María Candelaria Hernández Goya
Departamento de Ingeniería Informática y de Sistemas
Universidad de La Laguna
38200 San Cristóbal de La Laguna, Tenerife
España

Resumen · Abstract

Resumen

A lo largo de este documento se hará una breve introducción a la Criptografía y los principales conceptos y herramientas matemáticas utilizados para la descripción de los esquemas de encriptación. Principalmente, se presentará el concepto de criptografía homomórfica y se hará un breve repaso a su evolución. Además, se describirán los principales elementos criptográficos en los que se basa la seguridad de los esquemas homomórficos, tales como los problemas en retículos ideales.
Por último, se propondrá una implementación en el lenguaje Python del esquema de encriptación homomórfico BFV.

Palabras clave: *Criptografía Homomórfica – Retículos ideales – Esquema BFV.*

Abstract

This work will give a brief introduction to Cryptography and the main concepts and mathematical tools used for the description of encryption schemes. Most significantly, the concept of homomorphic cryptography will be presented and a brief review of its evolution will be made. Moreover, the main cryptographic elements on which the security of homomorphic schemes are based, such as the problems in ideal lattices, will be described.
Lastly, a Python implementation of the BFV homomorphic encryption scheme will be provided.

Keywords: *Homomorphic Cryptography – Ideal lattices – BFV scheme.*

Contenido

Introducción

En la antigüedad la Criptografía tenía como único fin la confidencialidad de los mensajes, esto es, la codificación de textos para restringir su contenido a las personas deseadas. En la actualidad esta misión se ha visto complementada por la autenticación, y la integridad de la información, entre otras propiedades de la Seguridad de la Información.

Este Trabajo Fin de Grado se enfocará únicamente en la confidencialidad de la información, es decir en la encriptación de esta. Para ello, la Criptografía moderna se basa en la teoría matemática y en la práctica de la informática. Los esquemas de encriptación se diseñan partiendo de primitivas criptográficas cuyas propiedades y seguridad está probada y desarrollan en torno a ellas algoritmos capaces de ofrecer confidencialidad en la información. La resistencia estos esquemas frente a ataques de terceras partes recae en los supuestos de complejidad computacional derivados de las primitivas elegidas.

Entre los supuestos de complejidad computacional utilizados por los esquemas de encriptación se encuentra los problemas en retículos ideales, que en la actualidad son uno de los principales pilares en los esquemas de encriptación capaces de afrontar el salto a la Computación Cuántica.

En este trabajo se hará una introducción a la Criptografía Homomórfica, la cual engloba al conjunto de esquemas de encriptación capaces de realizar operaciones con la información cifrada, de forma que el resultado descifrado sea equivalente a haber realizado las operaciones equivalentes con la información original.

Entre los esquemas de encriptación homomórficos se encuentra el esquema BFV (Bakerski/Fan-Vercauterense) que como se verá toma como supuesto de complejidad computacional el problema RWLE, que puede ser reducido a un problema en retículos ideales.

El documento que se presenta está estructurado de la siguiente manera:

- **Primer Capítulo.** Introducción a la Criptografía 1, donde se presentará la encriptación simétrica y asimétrica. También se describen conceptos básicos

de la Teoría de la Complejidad Computacional, que permitirán asociar los problemas matemáticos que se verán con la dificultad para romper un esquema. Por último haremos un breve repaso a la teoría de anillos y retículos.

- **Segundo Capítulo.** Criptografía Homomórfica 2, en el que se formalizará este concepto, se presentarán los problemas con retículos ideales y de aprendizaje sobre anillos que, junto a la Criptografía asimétrica, serán las principales primitivas criptográficas sobre las que se construirán los esquemas homomórficos que se tratarán. Además, se hará un breve repaso a la evolución de este campo.
- **Tercer Capítulo.** Esquema BFV 3. Para este capítulo se ha elegido uno esquema homomórfico, el cual se describe, y se realiza una implementación con fines didácticos en el lenguaje Python. Por último, se comenta otras librerías disponibles, para el trabajo con este y otros esquemas homomórficos

Los temas previamente mencionados se abordarán desde una perspectiva amplia y didáctica, introduciendo además otros esquemas de encriptación que facilitarán visualizar algunos conceptos interesantes y de actual relevancia como los retículos ideales o las técnicas de recifrado para esquemas algo homomórficos autoevaluable.

1

Introducción a la Criptografía

En este capítulo presentaremos varios conceptos básicos de Teoría de la Computación, Matemáticas y Criptografía que nos ayudarán a comprender los sistemas criptográficos y los mecanismos que estos utilizan.

Desde su inicio el fin principal de la Criptografía ha sido el desarrollar métodos para dificultar que agentes externos puedan acceder al contenido de una comunicación, como veremos más adelante este propósito original de la Criptografía se corresponde con lo que hoy en día conocemos como encriptación. En la actualidad la Criptografía ha adquirido una misión más amplia, sin limitarse solamente a la confidencialidad de las comunicaciones a través de la encriptación. Veamos a continuación una definición propuesta en [1] que nos esclarecerá en que consiste la Criptografía moderna.

Definición 1.1. *La Criptografía es el estudio de técnicas matemáticas relacionadas con aspectos de la seguridad de la información, tales como la confidencialidad, la integridad de datos, la autenticación de entidades y la autenticación del origen de los datos.*

El objetivo de la Criptografía es atender estos aspectos de la seguridad de la información de forma teórica y práctica. Algunas de las diferentes técnicas criptográficas o primitivas con las que podemos apoyarnos para lograr estos aspectos de la seguridad de la información son: son los esquemas de encriptación, los esquemas de firma digital o las funciones *hash*.

A lo largo de este capítulo y los siguientes se tratarán los esquemas de encriptación.

La encriptación tiene como objetivo mantener un conjunto de datos secreto o ininteligible para una tercera parte que no sea el emisor o receptor de una comunicación. La forma de dificultar o impedir el acceso al contenido original de un mensaje es mediante el cifrado de este, de forma que solo los participantes legítimos de la comunicación puedan descifrarlo para obtener el mensaje original. Al conjunto de algoritmos necesarios para realizar este proceso se los

denomina esquema de encriptación, a continuación definiremos formalmente en que consisten.

Definición 1.2. *Un esquema de encriptación $\mathcal{E}$ está formado por la tupla $(\mathcal{P}, \mathcal{C}, \mathcal{K}, \mathcal{ENC}, \mathcal{DEC})$ donde:*

- *$\mathcal{P}$, el espacio de texto plano, formado por los textos planos susceptibles de ser cifrados por el esquema.*
- *$\mathcal{C}$, el espacio de texto cifrado, que contiene textos cifrados o criptograma*
- *$\mathcal{K}$, el espacio de claves, formado por claves, que nos servirán para cifrar K_{Enc} o descifrar un texto K_{Dec}.*
- *$\mathcal{ENC} = \{Emc_k : k \in \mathcal{K}_{Enc}\}$, donde $Enc_k : \mathcal{P} \longrightarrow \mathcal{C}$ son las funciones de encriptación o cifrado, que permiten cifrar el texto plano.*
- *$\mathcal{DEC} = \{Dec_k : k \in \mathcal{K}_{Dec}\}$, donde $Dec_k : \mathcal{C} \longrightarrow \mathcal{P}$ son las funciones de desencriptación o descifrado, que permiten descifrar el texto cifrado.*

Además, $\forall e \in K_{Enc}\ \exists d \in K_{Dec} : Dec_d(Emc_e(m)) = m\ \forall m \in \mathcal{P}$.

Esta última condición que hemos dado en la definición implica que la función de encriptación y desencriptación no requieren necesariamente la misma clave para cifrar un texto plano, y obtener este mismo tras descifrarlo.

Se dirá que un esquema $\mathcal{E}$ tiene un nivel de seguridad de n-bits cuando sean necesarias al menos 2^n operaciones para romperlo.

1.1. Encriptación simétrica

La encriptación simétrica o de clave secreta, engloba al conjunto de esquemas de encriptación donde se utiliza la misma clave durante el proceso de encriptación y desencriptación. Por tanto, en estos esquemas la clave a utilizar debe ser negociada previamente entre el emisor y el receptor del mensaje, y esta debe mantenerse en secreto para impedir desencriptar el texto cifrado por un tercero.

En estos esquemas la confidencialidad del mensaje recae en última instancia en el secreto de la clave previamente compartida. Esto puede suponer un riesgo, puesto que la comunicación previa de la clave entre el emisor y el receptor no se encuentra encriptada y la filtración de la clave permitiría a un tercero desencriptar el texto cifrado.

Un ejemplo sencillo de un esquema simétrico es el cifrado Cesar.

Este esquema consiste en, dado un alfabeto, un conjunto de caracteres ordenado, de cuyos caracteres se compondrán los posibles elementos de nuestro espacio de texto plano $\mathcal{P}$, se podrá cifrar el texto plano sustituyendo cada carácter de este y descifrar realizando la sustitución inversa.

Para ello se escoge un número natural $k < n$, donde n es el número de elementos del alfabeto, k será la clave privada. Para cifrar el texto se sustituirá cada carácter por el que se encuentre k posiciones tras este en el alfabeto. El descifrado consistirá en realizar el proceso inverso, sustituir cada carácter de un texto cifrado por el que se encuentra k posiciones previo a este en el alfabeto.

Un caso particular de este esquema es conocido como ROT13 y es cuando se toma $k = 13$ puesto que en el alfabeto inglés donde $n = 26$ el algoritmo de cifrado es equivalente al de descifrado.

En la actualidad el sistema simétrico más relevante para la encriptación es el esquema AES *Advanced Encryption Standard* [2] estandarizado por el National Institute of Standards and Technology estadounidense en 2001.

1.2. Encriptación asimétrica

La encriptación asimétrica o de clave pública, recoge el conjunto de sistemas criptográficos en los que la clave de encriptación difiere de la clave de desencriptado. Estos esquemas permiten que la comunicación previa de la clave de encriptación no sea necesariamente secreta, puesto que el conocimiento por parte de un tercero de la clave de encriptación no compromete textos ya cifrados. En estos esquemas la confidencialidad de la comunicación recae principalmente en la clave de desencriptación, la cual será únicamente conocida por el receptor legítimo del mensaje y será la única que permita obtener el texto plano original.

En los esquemas asimétricos la clave de encriptación $e \in \mathcal{K}$ se la denomina clave pública, puesto que esta se pone en conocimiento del emisor sin mayor reparo en su confidencialidad. Por otra parte, la clave de desencriptado $d \in \mathcal{K}$ es llamada clave privada, aludiendo a que esta debe ser mantenida en secreto para evitar que alguien ajeno descifre el mensaje.

Veamos un par de ejemplos de sistemas asimétricos.

1.2.1. Esquema RSA

El esquema RSA, propuesto por R. Rivest, A. Shamir y L. Adleman en [3] es un ejemplo de uno de los sistemas de clave pública más utilizados en la actualidad. Este esquema se basa en el problema de factorización de números enteros, los cuales se obtendrán mediante el producto de dos números primos suficientemente grandes, dificultando de esta forma su factorización mediante fuerza bruta.

El esquema está formado por los algoritmos de generación de claves, cifrado y descifrado siguientes:

Generación de claves

1. Primero son elegidos dos números primos, p y q, y calculamos su producto $n = p \cdot q$.
2. Se elige d entero coprimo con $\psi(n)$, donde ψ denota la función de Euler. Al p y q primos se tiene que, $\psi(n) = \psi(pq) = \psi(p)\psi(q) = (p-1)(q-1)$. Se elige entonces un entero d tal que $MCD(d, (p-1)(q-1) = 1$.
3. Calculamos e como el inverso multiplicativo de d módulo $(p-1)(q-1)$ de forma que $e \cdot d \equiv 1 \pmod{(p-1)(q-1)}$.
4. Obtenemos como resultado la clave pública $k_p = (n, e)$ y la clave privada $k_s = (n, d)$.

Descubrir la clave privada consistirá en parte en factorizar n en números primos. En la publicación inicial del esquema se recomienda elegir p y q con al menos 100 dígitos, de esta forma n tendrá 200 dígitos, no obstante debido al avance de la informática y la mayor capacidad de cálculo disponible en la actualidad, se recomienda obtener n con un tamaño de al menos 2048 bits, es decir 617 dígitos decimales.

Cifrado

1. Codificamos nuestro texto plano M en un entero menor que n.
2. Calculamos el texto cifrado $c = m^e \pmod{n}$.

Descifrado

1. Calculamos $m = c^d \pmod{n}$. Donde c será el texto cifrado.
2. Convertimos el entero m obtenido en el texto plano M.

Como vemos la imposibilidad de descubrir la clave privada, y, por tanto, poder descifrar algún texto, en este esquema recae en la imposibilidad del atacante en calcular, de formar eficiente, la factorización de n. Esto le permitiría junto a la clave pública calcular nuestra clave privada.

En las implementaciones de este esquema suelen tenerse en consideración varios aspectos que aportan una mayor seguridad y eficiencia al esquema. Por ejemplo, aunque en [3] se elige d en la generación de claves, es más común elegir e de la misma forma y calcular d, ya que esto permite decidir un valor de e más bajo que reduce el cómputo al encriptar.

Un problema fundamental del esquema RSA plano, tal como se ha presentado, es que es vulnerable a ataques de texto plano elegido. Es decir, la encriptación de dos textos planos idénticos mediante la misma clave pública produce textos cifrados iguales, esto permite a una tercera parte comprobar si un texto cifrado coincide con un texto plano elegido.

Además, para ciertos valores de e pequeños o m para $m < n^{1/e}$ se tiene que $c = m^e \mod n = m^e$ y, por tanto, podríamos calcular $m = \sqrt[e]{c}$.

Ambos problemas de este esquema son solventados en las implementaciones de este mediante una función de relleno o *padding* que codifica el texto plano m añadiendo un factor no determinístico pero reversible antes de ser encriptado. Esto es conocido como RSA con relleno.

Definición 1.3. *Un esquema donde el texto cifrado sea indistinguible, entre sí, bajo un ataque de texto plano elegido, se dice que es semánticamente seguro [4].*

Las implementaciones del esquema RSA con relleno son semánticamente seguras.

1.2.2. Esquema ElGamal

Otro esquema asimétrico es el siguiente, presentado por T. Elgamal en [5] y basado en el esquema de intercambio de claves de Diffie-Hellman [6]. Este esquema se apoya en la dificultad del cálculo de logaritmos discretos sobre cuerpos finitos, y está compuesto por el algoritmo de generación de claves, encriptación y desencriptado siguientes.

Generación de claves

1. Primero elegimos un grupo cíclico $\mathcal{G}$ de orden p y un generador de este g.
2. Elegimos k_s nuestra clave privada en $0, 1, ..., p-1$.
3. Calculamos $k_p = g^{k_s}$ y obtendremos nuestra clave pública (p, g, k_p).

Cifrado

1. Codificamos nuestro texto plano m como un elemento de $M \in \mathcal{G}$.
2. Tomamos r de una distribución uniforme en $0, 1, ..., p-1$.
3. Calculamos $s = k_p^r$.
4. Obtenemos el texto cifrado como la tupla $(c_1 = g^r, c_2 = m \cdot s)$.

Descifrado

1. Sea $(c_1 = g^r, c_2 = m \cdot s)$ el texto cifrado, calculamos $s = c_1^{k_s} = g^{rk_s} = k_p^r$
2. Calculamos $s^{-}1$.
3. Obtenemos $m = c_2 \cdot s^{-1}$.

En la práctica, los esquemas de clave pública suelen utilizarse de forma mixta junto a los esquemas de clave secreta para asegurar la confidencialidad de la comunicación entre varias partes. Para ello la clave del esquema simétrico

es encriptada usando la clave pública, del esquema asimétrico, de las partes con las que deseamos comunicarnos de forma privada. La clave encriptada es entonces compartida con estos, quienes al tener la clave privada del esquema asimétrico podrán descifrarla. A partir de entonces se podrá mantenerse una vía de comunicación utilizando el esquema simétrico sin haber expuesto la clave de este.

El uso mixto de esquemas simétricos y asimétricos proporciona varias ventajas. Por una parte, el intercambio de la clave del esquema simétrico se realiza de forma confidencial gracias al sistema de clave pública, y por otra, salvo el mensaje encriptado inicial que incluye la clave del esquema simétrico, el resto de la comunicación es llevada a cabo bajo este.

Es de interés el uso mayoritario de un esquema simétrico durante la comunicación, ya que estos permiten reducir la cantidad de cálculos necesarios a la hora de encriptar y desencriptar los mensajes en comparación a un esquema asimétrico. La menor complejidad de estos esquemas permite agilizar la comunicación reduciendo el coste computacional.

1.3. Complejidad computacional

El área encargada del análisis de la complejidad y por consiguiente el coste en recursos, para resolver un problema computacional es la teoría de la complejidad computacional. La solución a estos problemas suele representarse a través de algoritmos.

A continuación se verán algunos conceptos clave, que nos permitirán discutir el coste computacional de los algoritmos subyacentes a un esquema, y la complejidad de los problemas matemáticos en los que estos se apoyan para ofrecer seguridad.

Definición 1.4. *Sean,* $f(x) : [0, \infty) \longrightarrow \mathbb{C}$ *y* $g : [0, \infty) \longrightarrow \mathbb{R}$*, diremos que g(x) es una cota superior asintótica, o O grande según la notación de Landau, si:*

$$\exists c > 0 \wedge \exists x_0 \geq 0 \ : \ |f(x)| \leq cg(x) \ \ \forall x \geq x_0 \tag{1.1}$$

Esta definición utilizada comúnmente en las series de Taylor para acotar el error al aproximar una función es usada también para dar una cota superior al tiempo de ejecución de un algoritmo en función de su entrada. Para nuestro fin $f(n)$ y $g(n)$ serán funciones de $\mathbb{N}$ en $\mathbb{N}$ en la definición anterior.

Describamos ahora algunos tipos de problemas computacionales de interés para nuestro discurso.

Definición 1.5. *Un problema de decisión es un problema computacional que independientemente del número de entradas tiene como respuesta, si o no.*

Un ejemplo de problema de decisión consiste en, dado un grafo G decidir si este es k-coloreable. Para tomar esta decisión podemos utilizar alguno de los algoritmos disponibles, y obtendremos como resultado de este un sí o un no.

Definición 1.6. *La clase P recoge al conjunto de problemas de decisión resolubles en tiempo polinomial por una máquina de Turing determinista.*

Definición 1.7. *La clase NP recoge al conjunto de problemas de decisión resolubles en tiempo polinomial por una máquina de Turing no determinista.*

Una manera alternativa de definir la clase NP es como el conjunto de problemas cuya solución es verificable en tiempo polinomial por una máquina de Turing determinística.

Además, diremos que un problema es NP-complejo (*NP-hard*) si para cada problema en NP podemos reducirlo en tiempo polinomial a este. Es decir, un problema NP-complejo tiene al menos la misma complejidad que los problemas más complejos en NP. Los problemas NP-complejos no pertenecen necesariamente a la clase NP.

En teoría de complejidad computacional se conoce como circuito un modelo matemático que describe como se obtiene una salida a una serie de cómputos y unos valores de entrada.

Definición 1.8. *Un circuito es un grafo dirigido acíclico $G = (V, E)$ cuyos vértices se denominan:*

- *Entradas, sí $gr^-(v_i) = 0 : v_i \in V$, donde $gr^-(v)$ denota el número de aristas incidentes en un vértice v. Son los valores de entrada para el cómputo.*
- *Salidas, sí $gr^+(v_i) = 0 : v_i \in V$, donde $gr^+(v)$ denota el número de aristas salientes en un vértice v. Son los resultados del cómputo.*
- *Puertas, son los vértices intermedios donde $gr^-(v_i) \neq 0 \wedge gr^+(v_i) \neq 0 : v_i \in V$. Estos llevan asociada una función de sus valores de entrada y cómputos precedentes.*

El tamaño de un circuito es el número de puertas que hay en este, y la profundidad de un circuito es la longitud del camino más largo empezando en una entrada y terminando en una salida.

1.4. Fundamentos matemáticos

Antes de adentrarnos en el funcionamiento de los sistemas criptográficos que detallaremos más adelante, es conveniente tener claro las estructuras algebraicas que han servido de herramienta para su construcción.

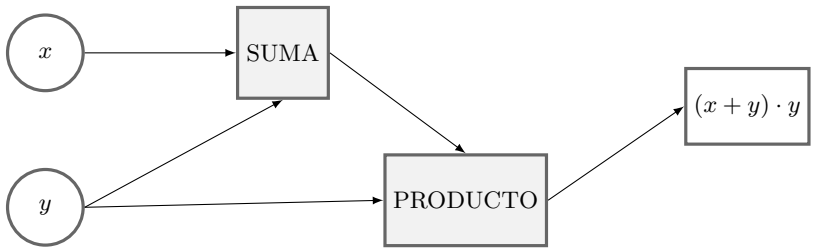


Figura 1.1: Ejemplo de circuito para el cómputo de $(x + y) \cdot y$, con tamaño 2 y profundidad 2.

1.4.1. Anillos, ideales y cuerpos

Definición 1.9. ***Anillo.*** *Un anillo $\mathcal{A}$ está formado por un conjunto A y dos leyes de composición interna en A denominadas comúnmente como suma, denotada por $+$, y multiplicación, denotada por $\cdot$. Cumpliendo $\mathcal{A} = (A, +, \cdot)$ las siguientes condiciones:*

- ***A1****. $(A, +)$ es un grupo conmutativo.*
- ***A2****. La multiplicación es asociativa, esto es:*

$$\forall a, b, c \in A \text{ se tiene que, } (a \cdot b) \cdot c = a \cdot (b \cdot c).$$

Además, existe un elemento neutro para la multiplicación, es decir:

$$\exists e \in A : \forall a \in A \; a \cdot e = e \cdot a = a.$$

- ***A3****. La multiplicación es distributiva sobre la suma:*

$$\forall a, b, c \in A \text{ se verifica que, } (a + b) \cdot c = a \cdot c + b \cdot c.$$

Si en el anillo $\mathcal{A}$ se cumple que:

$$\forall a, b, \in A \text{ se verifica que, } a \cdot b = b \cdot a,$$

entonces llamaremos a $\mathcal{A}$ anillo conmutativo o *abeliano*. Además, si en este anillo existe un elemento 1, tal que:

$$\exists 1 \in A : \forall a \in A \; 1 \cdot a = a \cdot 1 = a,$$

entonces $\mathcal{A}$ será un anillo conmutativo y unitario.En general este será el tipo de anillos con el que trabajaremos en adelante

Definición 1.10. ***Cuerpo.*** *Sea F un conjunto, y $+$, $\cdot$ dos leyes de composición interna, diremos que $\mathcal{F} = (F, +, \cdot)$ es un cuerpo si:*

- ***C1.*** *$(F,+)$ es un grupo abeliano.*
- ***C2.*** *$(F-\{0\},\cdot)$, donde 0 es el elemento neutro en $(F,+)$, es un grupo abeliano.*
- ***C3.*** *La multiplicación es distributiva sobre la suma, esto es*

$$\forall a,b,c \in F \text{ se verifica que, } (a+b)\cdot c = a\cdot c + b\cdot c.$$

Diremos que $\mathcal{S}=(S,+,\cdot)$ es un subanillo de $\mathcal{A}=(A,+,\cdot)$ si $\mathcal{S}$ es un anillo y $S \subseteq A$.

Definición 1.11. ***Ideal.*** *Un ideal I, o ideal por ambos lados, de un anillo $\mathcal{A}=(A,+,\cdot)$ es un subconjunto de A, tal que:*

- ***I1***. *$(I,+)$ es un subgrupo de A.*
- ***I2***. *Además,*

$$\forall a \in A \wedge \forall d \in I a\cdot d \in I \wedge d\cdot a \in I.$$

Si se verifica únicamente que $a\cdot d \in I$ en la última condición I será un ideal por la izquierda, y si únicamente $d\cdot a \in I$ entonces I será ideal por la derecha.

Sea $\mathcal{A}=(A,+,\cdot)$ un anillo e I un ideal, definimos la relación de equivalencia $\sim$:

$$a \sim b \leftrightarrow a-b \in I$$

Esta relación de equivalencia es compatible con las operaciones definidas en I. La clase de equivalencia, o la clase módulo I, de un elemento $a \in A$ es:

$$\overline{a} = a+I = (a+r : r \in I)$$

Definimos entonces el anillo cociente

Definición 1.12. ***Anillo cociente.*** *Sea $\mathcal{A}/I=(A/I,*,\circ)$, donde:*

$$A/I = \{\overline{a} = a+I : a \in A\}$$
$$\forall \overline{a},\overline{b} \in A/I \ \overline{a} * \overline{b} = \overline{a+b}$$
$$\forall \overline{a},\overline{b} \in A/I \ \overline{a} \circ \overline{b} = \overline{a\cdot b}$$

Hemos denotado las operaciones de este nuevo anillo de forma diferente para dejar clara su definición, no obstante es común en la literatura encontrarlas escritas como $+$ y $\cdot$ al igual que las operaciones en el anillo $\mathcal{A}$.

Sea $\mathcal{A}$ un anillo conmutativo y unitario, llamaremos polinomio en la indeterminada X en A a toda sucesión $(a_0,a_1,a_2,...)$ $a_i \in A$ donde todos sus elementos son nulos, salvo un número finito, y lo representaremos usualmente como $p(X)=a_0+a_1X+a_2X^2+...+a_iX^i+...+a_nX^n$. Definimos la suma de polinomios:

$$(a_0,a_1,a_2,...)+(b_0,b_1,b_2,...)=(a_0+b_0,a_1+b_1,a_2+b_2,...),$$

y el producto entre ellos de la forma siguiente:

$$(a_0, a_1, a_2, ...) \cdot (b_0, b_1, b_2, ...) = (c_0, c_1, c_2, ...), c_i = \sum_{j+k=i} a_j \cdot b_k$$

Además, dos polinomios serán iguales si las sucesiones mediante las que los hemos definido lo son, es decir, son igual término a término.

Definición 1.13. ***Anillo de polinomios.*** *Podemos construir el anillo de polinomios en una indeterminada en A,* $\mathcal{A}[X] = (A[X], +, \cdot)$ *donde:*

$$A[X] = \{p(X) = a_0 + a_1X + a_2X^2 + ... + a_iX^i + ... + a_nX^n,\ a_i \in A\},$$

considerando la suma y el producto de polinomios definidos anteriormente.

1.4.2. Espacios vectoriales y retículos

Sea V un espacio vectorial euclídeo de dimensión m, es usual encontrarnos en la necesidad de obtener una base ortogonal para un subespacio W de dimensión m de este.

Definición 1.14. *El proceso de ortogonalización de Gram-Schmidt consiste en dada una base* $B = \{b_1, ...b_m\}$ *de un subespacio vectorial* W*, determinar* $B^* = \{b_1^*, ...b_m^*\}$ *donde:*

$$b_i^* = b_i - \sum_{j=1}^{i-1} \mu_{i,j},\ \textit{donde,}\ \mu_{i,j} = \frac{\langle b_i, b_j^* \rangle}{\langle b_j^*, b_j^* \rangle}\ 1 \leq j < i \leq m \tag{1.2}$$

La principal estructura matemática que utilizaremos para explicar en que se basan el conjunto de sistemas criptográficos, que detallaremos, para garantizar que existe cierto grado de dificultad para descifrar textos por parte de terceros, son los retículos.

Definición 1.15. ***Retículo.*** *Dado un cuerpo* F *y un* F*-espacio vectorial de dimensión* n V*,* $B = \{b_1, ..., b_m\}$ *una base de un subespacio vectorial de este y* $\mathcal{A}$ *un anillo contenido en* F*, entonces el* $\mathcal{A}$ *retículo* Λ *en* V *generado por* B *es:*

$$\Lambda = \left\{ \sum_{i=1}^{m} a_i b_i \ :\ a_i \in A \right\} = \{B \cdot a : a \in A^m\} \tag{1.3}$$

Nosotros consideraremos de forma general el espacio vectorial $\mathbb{R}^n$, con B una base de este, y el anillo $\mathbb{Z}$ para construir el retículo $\Lambda = \{B \cdot a : a \in \mathbb{Z}^n\}$. Por tanto, nuestra base B será una matriz cuadrada de rango n,y diremos en este caso que el retículo es de dimensión máxima.

Observamos que el retículo no es más que un subgrupo aditivo de $\mathbb{R}^n$ formado por la restricción a coeficientes enteros de la combinación lineal de la base B.

De esta forma un retículo Λ puede estar generado por diferentes bases. No obstante, no cualquier base del espacio vectorial, $\mathbb{R}^n$, es necesariamente una base de Λ. Denotaremos al retículo por $\Lambda(B)$ cuando queramos hacer énfasis en la base.

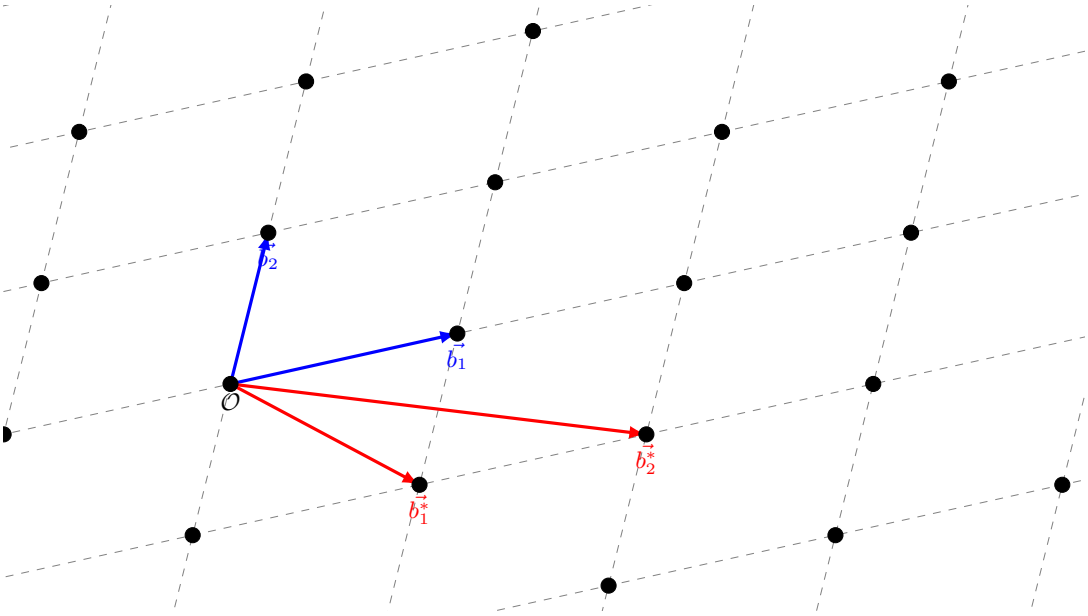


Figura 1.2: Ejemplo de retículo Λ en $\mathbb{R}^2$, generado tanto por $B = (\vec{b_1}, \vec{b_2})$ como por $B^* = (\vec{b_1^*}, \vec{b_2^*})$.

Además, se tiene que si B y B^* son bases que generan Λ, entonces $B = U \cdot B^*$ para cierta matriz U unimodular, esto es $det(U) = \pm 1$.

Para un retículo Λ podemos ver que $|det(B)|$ es invariante con respecto a la base B que elijamos tal que $\Lambda = \Lambda(B)$ puesto que si B^* es otra base, se verifica que $det(B) = det(U) \cdot det(B^*) = \pm det(B^*)$ puesto que U es unimodular. Llamaremos discriminante de Λ a:

$$\Delta(\Lambda) = |det(B)| \tag{1.4}$$

Para una base B con $rango(B) = n$ máximo de un retículo Λ, el valor del discriminante se corresponde geométricamente con el volumen del paralelepípedo fundamental $\mathcal{P}(B) = \{x = \sum_{i=1}^{n} x_i b_i : -\frac{1}{2} \leq x_i < \frac{1}{2}\}$ del retículo, o equivalentemente con el de la región de Voroni de cualquier elemento del retículo.

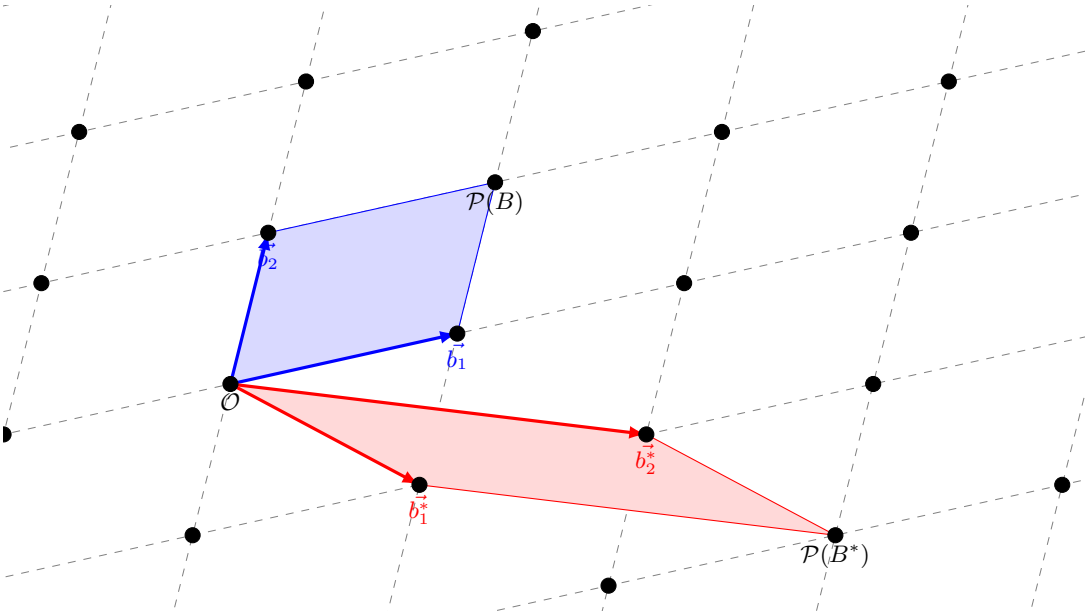


Figura 1.3: Ejemplo de paralelepípedo fundamental trasladado de $\Lambda(B)$ y $\Lambda(B^*)$. Notamos que $vol(\mathcal{P}(B^*)) = vol(\mathcal{P}(B)) = \Delta(\Lambda)$.

Dentro de los problemas con retículos que presentaremos en el siguiente capítulo veremos que dado un retículo, será interesante conseguir una base lo más ortogonal posible.

En un espacio vectorial podemos utilizar el proceso de ortogonalización de Gram–Schmidt para obtener una nueva base ortogonal; sin embargo, como se ha comentado, una base de Gram-Schmidt no tiene por qué estar en el retículo y, por tanto, no generar este.

Una forma de comparar la ortogonalidad de diferentes bases es mediante el defecto de ortogonalidad $\delta(B)$.

$$\delta(B) = \frac{\prod_{i=1}^{n} \|b_i\|}{\Delta(\Lambda)} \tag{1.5}$$

De esta forma, para una base ortogonal B tendremos que $\delta(B) = 1$, ya que verificara que el producto de la norma de los vectores de la base, la longitud de las aristas de nuestro paralelepípedo n-dimensional, es igual al volumen que este encierra. Así cuanto más cercano $\delta(B)$ este a 1 más ortogonal será nuestra base.

Dado un retículo λ y una base B de este, se puede tener la necesidad de obtener una base B^* más ortogonal. El algoritmo LLL presentado por A. K. Lenstra, H. W. Lenstra, y L. Lovász en [7] permite obtener una base, que llamaremos LLL reducida, con un defecto de ortogonalidad menor en tiempo polinomial.

Definición 1.16. *Una base B de un retículo Λ de dimensión máxima n se dice δ-LLL reducida si:*

1. *$\forall i, j : 1 \leq j < i \leq n$ se tiene que, $|\mu| \leq \frac{1}{2}$.*
2. *Además $\forall i : 1 < i \leq n \ \|b_i^*\| \geq (\delta - \mu_{i,i-1}^2)\|b_{i-1}^*\|$.*

Donde $B^* = \{b_1^*, ..., b_n^*\}$ es la base que obtenemos mediante el proceso de ortogonalización de Gram–Schmidt 1.14, y los $\mu_{i,j}$ calculados en este.

El algoritmo utilizado para obtener una base LLL-reducida en tiempo polinomial parte de una base B y consiste en:

1. Se obtiene una base ortogonal $B^* = \{b_1^*, ..., b_n^*\}$ mediante el proceso de Gram–Schmidt.
2. Para $2 \leq i \leq n$ y para cada $1 \leq j \leq i-1$ se toma b_i por

$$b_i - c_{i,j}b_j, \ c_{i,j} = \lceil \frac{\langle b_i, b_j^* \rangle}{\langle b_j^*, b_j^* \rangle} \rfloor$$

.
3. Si tras esto se tiene algún i tal que $\|b_i^*\| < (\delta - \mu_{i,i-1}^2)\|b_{i-1}^*\|$ intercambiamos b_i y b_{i-1} en nuestra base y volvemos al primer paso.
4. Si tras la comprobación anterior no obtenemos ningún i, la base resultante B será δ-LLL reducida.

De esta forma es posible obtener una base de Λ más ortogonal que nuestra base inicial.

Este algoritmo es de interés a la hora de analizar la seguridad de criptosistemas basado en retículos, pues como se verá obtener una base lo más ortogonal posible de cierto retículo podría permitir descifrar texto cifrado por terceros.

Una base alternativa de un retículo es la que se obtiene al considerar la forma normal de Hermite de una base B.

Definición 1.17. *Se dice que una matriz H es una forma normal de Hermite de una matriz B de dimensión $n \times n$ si existe U matriz unimodular tal que $H = UB$ y H verifica:*

- *H es triangular inferior, esto es $h_{i,j} = 0 sii < j$.*
- *Los elementos de la diagonal principal son positivos $h_{i,i} > 0$.*
- *Si $i > j$, $h_{i,j} \in [-b_{j,j}/2, b_{j,j}/2)$.*

La forma normal de Hermite H de una base B es también una base del retículo $\Lambda(B)$, pues por construcción de H esta es una transformación de B por una matriz unimodular. Como se verá es usual considerar H en los esquemas basados en retículos.

Definición 1.18. *El retículo dual Λ^* de un retículo Λ en un subespacio de $\mathbb{R}^n$ es*

$$\Lambda^* = \{\vec{x} \in gen(\Lambda) \ : \ \langle \vec{x}, \vec{y} \rangle \in \mathbb{Z} \forall \vec{y} \in \Lambda\}. \tag{1.6}$$

Para las bases B de rango máximo se tiene que $(B^T)^{-1}$ es una base de Λ^*. Una consecuencia de esto es que $\Delta(\Lambda^*) = det((B^T)^{-1}) = \frac{1}{det(B)}$. Además, el defecto de ortogonalidad del retículo dual es $\delta(B^*) = det(B) \prod_{i=1}^{n} \|b_i^*\|$, donde b_i^* son los elementos de la base del dual, las columnas de B^*.

Dada una norma vectorial $\|\cdot\|$ en V llamaremos mínimo del retículo Λ y denotaremos por $\lambda_1(\Lambda)$ al mínimo valor de la norma de un vector de λ, es decir:

$$\lambda_1(\Lambda) = \text{mín}(\|\vec{x}\| : \ \vec{x} \in \Lambda). \tag{1.7}$$

Otra noción que necesitaremos más adelante será la de los mínimos sucesivos del retículo de dimensión n Λ, $\lambda_i(\Lambda)$.

Definición 1.19. *Para cualquier retículo Λ de dimensión n. Sea $1 \leq i \leq n$ entero, y una norma $\|\cdot\|$ entonces: $\lambda_i(\Lambda) = min(r \in \mathbb{R}^+$ tal que $\overline{B(0,r)}$ contiene exactamente i vectores de Λ linealmente independientes.*

Dos resultados que son de bastante interés para la Criptografía basada en retículos, ya que nos proporcionan cotas para los mínimos de un retículo son los siguientes.

Teorema 1.20 (Teorema de Minkowski). *Para cualquier retículo Λ de dimensión n máxima, se tiene que*

$$\lambda_1(\Lambda) \leq \sqrt{n} \cdot \Delta(\Lambda)^{1/n}. \tag{1.8}$$

Teorema 1.21 (Teorema de Transferencia de Banaszczyk). *Para cualquier retículo Λ de dimensión n, se tiene que*

$$1 \leq \lambda_1(\Lambda) \cdot \lambda_n(\Lambda^*) \leq n. \tag{1.9}$$

Donde Λ^* denota el dual de Λ.

Mediante ambos teoremas es posible acotar el valor de los mínimos sucesivos de Λ.

Retículos ideales

Para la construcción de un retículo se puede plantear considerar el anillo $\mathcal{A} = \mathbb{Z}x/f(x)$, con $f(x)$ un polinomio mónico irreducible de grado n con coeficientes en $\mathbb{Z}$, en el cuerpo $\mathbb{Q}[x]/f(x)$. Los elementos de $\mathcal{A}$ son los polinomios con

coeficientes en $\mathbb{Z}$ de grado $n-1$, cuyos coeficientes se pueden ver como vectores de $\mathbb{Z}^n$. Se considerará, por simplicidad, $f(x) = x^n + 1$.

En el anillo $\mathcal{A}$ un ideal principal I, es decir un ideal generado por un único elemento $p(x) = p_0 + p_1 x + \ldots + p_{n-1}x^{n-1} \in \mathcal{A}$, al ser cerrado para la suma en el anillo es a la vez un retículo cuya base está formada por $\vec{p^i}$ los vectores cuyos elementos son los coeficientes de $p(x) \cdot x^i \mod f(x) : \ i \in [0, n-1]$. A este retículo se le denomina retículo ideal, haciendo referencia a las estructuras algebraicas que subyacen en él, y a la base P anterior se la llama base rotacional del retículo ideal.

La base P tendrá la forma:

$$P = \begin{bmatrix} p_0 & -p_{n-1} & -p_{n-2} & \cdots & -p_1 \\ p_1 & p_0 & -p_{n-1} & \cdots & -p_2 \\ p_2 & p_1 & p_0 & \cdots & -p_3 \\ \vdots & & & \ddots & \\ p_{n-1} & p_{n-2} & p_{n-3} & \cdots & p_0 \end{bmatrix} \tag{1.10}$$

Para un ideal principal I como se ha definido se tiene que, su inverso $I^{-1} = \{\vec{w} \in \mathbb{Q}[x]/f(x) : \ \forall \vec{v} \in I, \vec{v} \times \vec{w} \in \mathcal{A}\} = (\vec{q})$, donde $\vec{q}$ son los coeficientes de $q(x) = p^{-1}(x)$ en dicho cuerpo.

2

Criptografía Homomórfica

En este capítulo trataremos en que consisten los sistemas criptográficos homomórficos, haremos un breve repaso por su evolución y presentaremos uno de los principales problemas en los que se apoyan varios de estos sistemas en la actualidad.

2.1. Introducción

La criptografía homomórfica es la rama de la criptografía que engloba los esquemas de encriptación homomórficos. Estos sistemas son aquellos que permiten realizar un conjunto de operaciones sobre el texto cifrado, de forma que el resultado sigue estando cifrado y al desencriptarlo obtendremos el resultado de esta operación en texto plano.

Es decir, siguiendo la notación del capítulo previo, para un esquema de encriptación $(\mathcal{P}, \mathcal{C}, \mathcal{K}, \mathcal{E}, \mathcal{D})$ y una operación binaria interna $F_{\mathcal{P}}$ en $\mathcal{P}$ se tiene $F_{\mathcal{C}}$ en $\mathcal{C}$ verificando:

$$F_{\mathcal{P}}(m_1, m_2) = Dec_d(F_{\mathcal{C}}(Enc_e(m_1), Enc_e(m_2))) : m_1, m_2 \in \mathcal{P}, d \in K_{Dec}, e \in K_{Enc}. \tag{2.1}$$

En general, esta operación se corresponde con la suma o la multiplicación.

A los esquemas de encriptación que permiten una única operación verificando 2.1, los llamaremos esquemas parcialmente homomórficos *(PHE)*.

Por otra parte, a los esquemas que permiten dos operaciones, suma y multiplicación, cumpliendo (**??**), se los denomina esquemas completamente homomórficos *(FHE)*.

En el caso de que un esquema de encriptación permita dos operaciones verificando (**??**) para circuitos de una profundidad determinada, es decir, esta

característica deje de ser cierta tras un número finito de operaciones consecutivas, diremos que estamos ante un esquema algo homomórfico *(SHE)*. Algunos esquemas de este tipo nos permiten predefinir el número de operaciones consecutivas límite n para seguir manteniendo esta propiedad, estableciendo los parámetros del esquema de forma acorde. A este conjunto se los denomina esquemas homomórficos nivelados y son equivalentes a un esquema completamente homomórfico para circuitos de profundidad n.

Propiedad homomórfica del esquema RSA

Un ejemplo de esquema parcialmente homomórfico es el esquema RSA, que hemos visto en 1.2.1. En este esquema, si no aplicamos relleno al texto plano, la multiplicación es homomórfica y podemos comprobar como el producto de dos números cifrados es equivalente al cifrado del producto de estos mismos.

Dada una clave pública $k = (e, n)$:

$$\begin{aligned} Enc_k(m_1) \cdot_{\mathcal{C}} Enc_k(m_2) = {m_1}^e \mod n \cdot_{\mathcal{C}} {m_2}^e \mod n = \\ = (m_1 \cdot m_2)^e \mod n = Enc_k(m_1 \cdot m_2) \end{aligned} \tag{2.2}$$

No obstante, en la práctica no es seguro utilizar el esquema RSA sin relleno, ya que como comprobamos anteriormente es vulnerable a diferentes ataques que dejan expuesto el texto plano original.

Propiedad homomórfica del esquema ElGamal

Otro esquema parcialmente homomórfico es el que hemos introducido en 1.2.2. El esquema ElGamal también es homomórfico con respecto a la multiplicación únicamente, puesto que dada una clave pública para este esquema $k = (G, ord(G), p, q)$, se verifica:

$$\begin{aligned} Enc_k(m_1) \cdot_{\mathcal{C}} Enc_k(m_2) = (p^{r_1}, m_1 \cdot q^{r_1}) \cdot_{\mathcal{C}} (p^{r_2}, m_2 \cdot q^{r_2}) = \\ = (p^{r_1+r_2}, (m_1 \cdot m_2) \cdot q^{r_1+r_2}) = Enc_k(m_1 \cdot m_2). \end{aligned} \tag{2.3}$$

Donde r_1 y r_2 son elegidos de forma uniforme en $(0, 1, ..., ord(G) - 1)$.

2.2. Criptografía basada en retículos

En esta sección se introducirán los principales problemas con retículos, los cuales, sirviendo como supuestos de complejidad computacional, son utilizados como primitivas criptográficas en la construcción de los principales esquemas homomórficos.

Además, se presentará el esquema GGH 2.2.4, que, aún sin ser homomórfico, permitirá entender de una forma visual el uso de los problemas con retículos como primitiva criptográfica.

2.2.1. Problemas con retículos

De manera ilustrativa consideraremos ejemplos en $(R)^2$ no obstante, como veremos en la práctica es necesario trabajar en dimensiones superiores.

Problema del vector más corto

El problema del vector más corto o *Shortest Vector Problem, SVP* consiste en, dada una norma $\|\cdot\|$, encontrar el vector más corto en un retículo.

Definición 2.1. *Sea un retículo Λ, y $\|\cdot\|$ una norma vectorial. Entonces, el problema del vector más corto consiste en encontrar un vector $\vec{x} \in \Lambda$, $\vec{x} \neq 0$ tal que:*

$$\forall \vec{y} \in \Lambda - \{0\} \ |\vec{x}| \leq |\vec{y}| \tag{2.4}$$

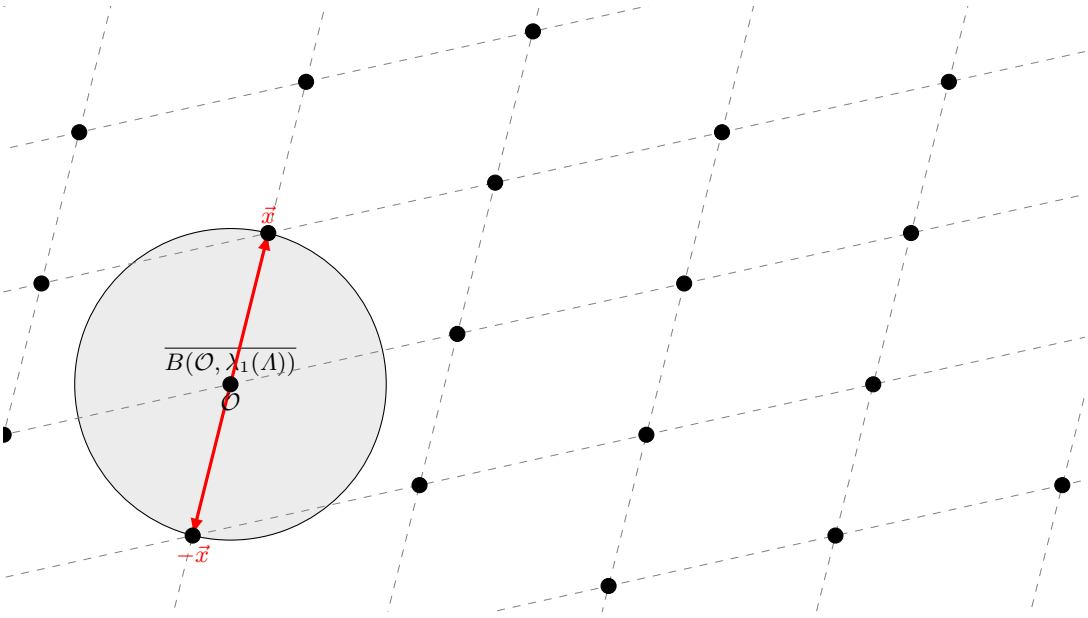


Figura 2.1: En este ejemplo tanto $\vec{x}$ como $-\vec{x}$ son solución para el SVP en este retículo en $\mathbb{R}^2$.

Notamos que, $\lambda_1(\Lambda) = min(|\vec{y}| : \ \vec{y} \in \Lambda) = |\vec{x}|$. Podemos considerar el problema del vector más corto aproximadamente, si dada una constante $\gamma \in \mathbb{R}^+$, y bajo el enunciado anterior, buscamos $\vec{x} \in \Lambda - \{0\}$ verificando:

$$|\vec{x}| \leq \gamma \lambda_1(\Lambda) \tag{2.5}$$

Problema del vector mas cercano

El problema del vector más cercano o *Closest Vector Problem, CVP* consiste en dado un vector en V, el espacio vectorial en el que se define Λ, encontrar el vector del retículo Λ que diste menos de él.

Definición 2.2. *Sea el retículo Λ en el espacio vectorial V y el vector $\vec{x} \in V$, Entonces el problema de vector más cercano consiste en encontrar $\vec{y} \in \Lambda$ tal que:*

$$\forall \vec{z} \in \Lambda \ \|\vec{x} - \vec{y}\| \leq \|\vec{x} - \vec{z}\| \tag{2.6}$$

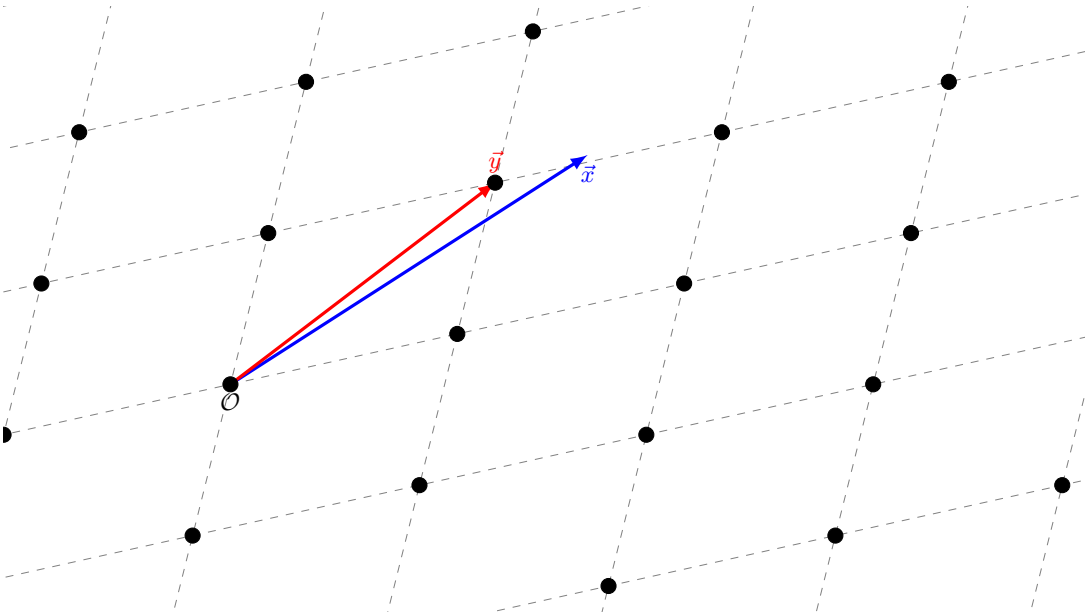


Figura 2.2: Ejemplo de solución para el CVP en $\mathbb{R}^2$

Al igual que en el caso del SVP podemos considerar el problema del vector más cercano aproximadamente. Sea $\gamma \in \mathbb{R}^+$, buscaremos en este caso $\vec{y} \in \Lambda - \{0\}$ verificando:

$$\forall \vec{z} \in \Lambda \ \|\vec{x} - \vec{y}\| \leq \gamma \|\vec{x} - \vec{z}\| \tag{2.7}$$

2.2.2. Aprendizaje con errores

El problema de aprendizaje con errores *(LWE)* fue descrito por primera vez por O. Regev en [8]. Una de las maneras alternativas en las que lo presenta, como el problema de recuperar códigos lineales aleatorios, es la siguiente:

Definición 2.3. *Sea $n \in \mathbb{N}$, y un $m \in \mathbb{N}$ arbitrario, $\vec{s} \in \mathbb{Z}_p^n$. Entonces dada $M \in \mathbb{Z}_p^{m \times n}$, y el vector $\vec{b} = [M\vec{s} + \vec{e}]_q$ donde $\vec{e} \in \mathbb{Z}_p^m$ es un vector formado por m muestras independientes de una distribución probabilística fijada en Z_p.*

El problema de búsqueda de aprendizaje con errores consiste en obtener $\vec{s}$ *conociendo únicamente* $M y \vec{b}$*, es decir,* m *duplas* $(\vec{a_i} = \vec{m}_i, b_i)$

En su desarrollo, Regev considera en particular $\vec{e}$ tomado de una distribución normal discreta, o redondeada al entero más cercano, tal que $e_i \sim N(0, \frac{a^2}{2\pi}) = D_a$.

2.2.3. Aprendizaje con errores sobre anillos

El problema computacional de aprendizaje con errores sobre anillos *(RLWE)* fue descrito por Lyubashvsky, Peikert y Regev en [9]. El aprendizaje con errores sobre anillos se presenta como una variación algebraica del aprendizaje con errores que permite mejorar la eficiencia en los cálculos.

Definición 2.4. *Sea* $n = 2^k$ *para un* $k \in \mathbb{N}$*, y* $q \in \mathbb{N}$*. Se consideran* $s(x) \in \mathcal{A}_q = \mathbb{Z}_q[x]/(x^n + 1)$ *y* m *muestras* $e_i(x)$ *independientes de una distribución* χ *en* $\mathcal{A} = \mathbb{Z}[x]/(x^n + 1)$ *acotada y* m *muestras independientes* $a_i(x) \longleftarrow U(\mathcal{A})$

El problema de búsqueda de aprendizaje con errores sobre anillos consiste en obtener $s(x)$ *partiendo de las* m *duplas* $(a_i(x), b_i(x) = [a(x) \cdot s(x) + e_i(x))]_q$.

Siguiendo la notación de este problema se presenta el problema de decisión asociado.

Definición 2.5. *El problema de decisión de aprendizaje con errores sobre anillos (D-RLWE) consiste en, muestreada un número arbitrario de duplas* $(a_i(x), b_i(x))$ *según se indica en 2.4, decidir si los* $b_i(x)$ *son una muestras de* $U(\mathcal{A})$ *o fueron construidos como* $[a_i(x) \cdot s(x) + e_i(x))]_q$.

Como se puede apreciar, en comparación con el LWE, cada muestra de $b(x)$ proporciona n muestras sobre Z_q correspondientes a cada coeficiente, de esta manera, en la mayoría de las aplicaciones del problema LWE una muestra de $(a(x), b(x))$ para el problema RLWE puede tomarse como n muestras $(\vec{a}, b_i)$ del LWE, reduciendo el costo de almacenaje de las m duplas. Por otra parte, el cálculo de forma polinomial de $b(x)$ proporcionando n muestras de b_i mediante $O(n \log n)$ operaciones escalares es más eficiente que el cálculo equivalente en el LWE.

En el artículo [9] también se prueba la complejidad de los problemas de búsqueda y de decisión RLWE mediante una reducción del peor caso de este al problema SVP sobre retículos, que se ha presentado en 2.1.

Aunque esta prueba considera ciertas restricciones en el q considerado, en [10] se muestra como estas restricciones son innecesarias para mantener la complejidad del problema y, por tanto, la seguridad de los esquemas basados en este, permitiendo también adaptar con mayor facilidad esquemas basados en el problema LWE, que carecían de estas restricciones, a RLWE.

En la publicación original de este problema se presenta también un esquema de encriptación en el cual se basará el esquema BFV que explicaremos en 3.1.

2.2.4. Esquema GGH

Hacemos una mención al esquema de encriptación GGH publicado en [11] por O. Goldreich, S. Goldwasser y S. Halevi en 1997, junto al esquema de firmas homónimo, pues es un ejemplo de criptosistema basado en retículos, que aún sin ser homomórfico, nos puede servir de base para entender los esquemas homomórficos que comentaremos.

Este esquema de encriptación se basa en la capacidad de diferentes bases de un retículo de dar aproximaciones mejores o peores de vectores arbitrarios en $\mathbb{R}^n$. El problema de encontrar el vector del retículo más cercano a este punto arbitrario es un caso particular del CVP como veremos.

Mediante los algoritmos que veremos el esquema escoge una base "buena" R que será nuestra clave privada y una base "mala" B que formara parte de la clave pública del mismo retículo Λ, de forma que tomando un mensaje m codificado como las coordenadas en la base B de un vector del retículo y añadiendo un pequeño error no determinístico obtenemos un vector de $\mathbb{R}^n$ que no se encuentra en el retículo. Mediante la base R que buscaremos con el menor defecto de ortogonalidad del dual, será posible identificar correctamente el vector del retículo en el que hemos codificado m. Otra base del retículo, como B, no nos permitirá obtener el vector del retículo más cercano correctamente. Encontrar el vector del retículo más cercano a nuestro vector cifrado en $\mathbb{R}^n$ es un caso particular del CVP.

Se presenta a continuación los algoritmos que forman este esquema.

Generación de claves

1. Elegimos la dimensión del retículo que usaremos, n.
2. Tomamos una matriz R de una distribución uniforme en $GL_n(\mathbb{Z})$, y consideramos el retículo $\Lambda(R)$.
3. A partir de la base R obtenemos una base B con un mayor defecto de ortogonalidad. Para ello, los autores proponen alterar cada r_i vector de la base en R sumando una combinación lineal del resto. De forma que $b_i = r_i + \sum_{\substack{j=1 \\ j \neq i}}^{n} a_j r_j$ donde $a_j \in \{-1, 0, 1\}$. Iterando este procedimiento $2n$ veces podemos obtener una base B capaz de prevenir que el algoritmo LLL pueda obtener una reducción.
4. Elegimos $\sigma \in \mathbb{N}$ tal que $\sigma < \frac{1}{2 \max_{1 \leq i \leq n} \|r_{i\cdot}^*\|}$ donde $\|\cdot\|$ es la norma L_1 y $r_{i\cdot}^*$ es la fila i de R^{-1}.

5. Finalmente, obtenemos nuestra base privada R y pública (B, σ). Y almacenamos para facilitar el cómputo posterior las matrices R^{-1}, y $T = B^{-1}R$.

Cabe destacar que los autores recomiendan considerar $n >= 100$ pues experimentalmente son capaces de conseguir bases con muy poco defecto de ortogonalidad para $n <= 80$.

Alternativamente, al procedimiento para generar una base pública indicado en el artículo original, D. Micciancio propone en [12] escoger B como la forma normal de Hermite de R.

Cifrado

1. Codificamos nuestro mensaje m en un vector $\vec{m} \in \mathbb{Z}^n$, que determinara las coordenadas de un elemento del retículo $\vec{v} = B\vec{m}$
2. Consideramos $\vec{e}$ formado por n entradas tomadas de una distribución uniforme en $\{-\sigma, \sigma\}$.
3. Calculamos el texto cifrado $\vec{c} = B\vec{m} + \vec{e}$

En general el vector $\vec{e}$ puede ser cualquier perturbación a $\vec{v}$, no obstante se debe tener cuidado con la distribución de este.

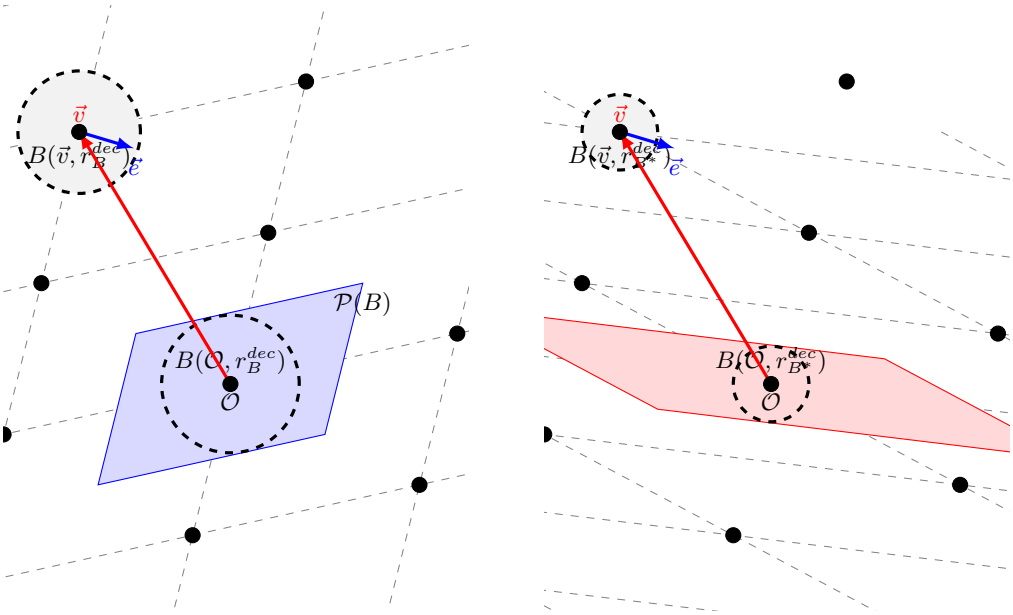
Considerar una varianza muy grande conllevará aumentar la probabilidad de descifrar incorrectamente, no obteniendo el texto plano original. Por otra parte, tomar una distribución con una baja varianza puede afectar a la seguridad del esquema.

Descifrado

1. Partiendo del vector $\vec{c}$ en $\mathbb{R}^n$, se calcula $\vec{m} = T\lfloor \mathbb{R}^{-1}\vec{c} \rceil$ donde $\lfloor \cdot \rceil$ denota la técnica de redondeo de Babai.
2. Se decodifica $\vec{m}$ en el mensaje m.

Un ejemplo de la diferencia entre desencriptar $\vec{v}$ utilizando una base privada B, y una base pública B^*, tomadas siguiendo el ejemplo de 1.2, puede verse en 2.3. La base B con un menor defecto de ortogonalidad tiene un mayor radio de descifrado r^{dec}, este es el radio de la circunferencia inscrita en el paralelepípedo fundamental $\mathcal{B}$. En general, para el caso n dimensional podemos hablar de $B(\mathcal{O}, r^{dec})$ inscrita en la región de dimensión n análoga formada por los elementos de una base.

En 2.3a al ser la base lo suficientemente ortogonal para que la circunferencia de radio r_B^{dec} inscrita en $\mathcal{P}(B)$ permita que $\|\vec{e}\| \leq r_B^{dec}$. De esta forma, la base B permitirá obtener el vector original $\vec{v}$ que codifica el mensaje, ya que $\vec{c}$ se



(a) Para una base B donde $\|\vec{e}\| \leq r_B^{dec}$, podremos descifrar $\vec{v}$.

(b) Para una base B^* donde $\|\vec{e}\| > r_{B^*}^{dec}$, no podremos asegurar que el desencriptado sea correcto.

Figura 2.3: Ejemplo del radio de decodificación r^{dec} para dos bases B y B^*, tal que B es más ortogonal que B^*.

encuentra en la región $B(\vec{v}, r_B^{dec})$, representando $\vec{c}$ en esta base y redondeando como se ha indicado.

Por otra parte, en 2.3b se tiene la circunferencia inscrita en $\mathcal{P}(B^*)$ de radio $r_{B^*}^{dec}$. En este caso $\|\vec{e}\| > r_{B^*}^{dec}$ y, por tanto, $\vec{c} \notin B(\vec{v}, r_{B^*}^{dec})$ encontrar en este caso el elemento de Λ más cercano a $\vec{c}$ puede suponer en el peor caso resolver el CVP.

En la práctica no se recomienda el uso del esquema GGH, pues este no es semánticamente seguro. Para una codificación $\vec{m}$ de un texto plano, se puede comprobar calculando $\vec{c} - B\vec{m}$ si $\vec{c}$ es corresponde con el texto cifrado del texto plano m.

Además, en 1999 P. Nguyen publica un análisis de este criptosistema [13], en el que explica como reducir el CVP a un caso particular de este. De manera abreviada, Nguyen describa como se puede determinar un número reducido de soluciones para $\vec{c} + \vec{s} \equiv B\vec{m} \mod 2\sigma$ con $\vec{s} = (\sigma, ..., \sigma)$ en $\vec{m} \mod 2\sigma$, y a través de este reducir el CVP donde la longitud máxima del error es $\sigma\sqrt{n}$ a resolver el CVP donde $\|\vec{e}\| \leq \sqrt{n}/2$. Permitiendo aplicar algoritmos que resuelven este caso particular del CVP, y obtener así $\vec{m}$.

2.3. Esquemas completamente homomórficos

Gran parte de los esquemas completamente homomórficos se basan en los problemas con retículos que hemos discutido, y en el trabajo de C. Gentry, quien en [14] propone una manera de construir estos esquemas partiendo de esquemas algo homomórficos capaces de evaluar su propio circuito de descifrado.

A continuación se verá como obtener un esquema completamente homomórfico a partir de un esquema algo homomórfico según Gentry, quien a partir de esto construye un esquema basado en retículos ideales. Tras esto se hará un repaso por otros esquemas homomórficos basados en estas ideas y en los problemas que se han visto en la sección anterior.

2.3.1. Construcción de esquemas completamente homomórficos

En la primera parte de [14] se describe de forma teórica la siguiente forma para construir un esquema completamente homomórfico partiendo de un esquema homomórfico autoevaluado.

Definición 2.6. *Se dice que un esquema homomórfico $\mathcal{E}$ es autoevaluado o bootstrappable si, para $\mathcal{C}_\mathcal{E}$ el conjunto de circuitos que es capaz de evaluar $\mathcal{E}$ homomórficamente, se tiene que:*

$$D_\mathcal{E}(\Gamma) \subseteq \mathcal{C}_\mathcal{E} \tag{2.8}$$

Donde Γ denota al conjunto de puertas en $\mathcal{P}$, el espacio de texto plano, y $D_\mathcal{E}(\Gamma)$ denota al conjunto de circuitos formado por las puertas $g \in \Gamma$ que toman como entradas el circuito de descifrado de $\mathcal{E}$.

A partir de $\mathcal{E}$, se construye para la profundidad del circuito en $\mathcal{P}$ deseada $d \geq 1$ el esquema $\mathcal{E}^d$, capaz de evaluar homomórficamente circuitos de profundidad $\delta \leq d$, con sus algoritmos de generación de claves, cifrado, descifrado y evaluación asociados. Este esquema se conforma por $d+1$ copias del esquema $\mathcal{E}$ de forma que:

- Se generan $d+1$ pares de claves públicas $k_{p,i}$ y secretas $k_{s,i}$ utilizando la generación de $\mathcal{E}$. Se cifra con el algoritmo de $\mathcal{E}$ $k_{s,ij}$, la representación de $k_{s,i}$ en $\mathcal{P}$, usando la clave $k_{p,i-1}$. Para el esquema $\mathcal{E}^\delta$ se tiene la clave privada $k_s^\delta = k_{s,0}$ y la clave pública $k_p^\delta = (k_{p,i}\ i \in [0,\delta], Enc_\mathcal{E}(k_{p,i-1}, k_{s,ij})\ i \in [1,\delta])$.
- Para un texto plano $p \in \mathcal{P}$, $Enc_{\mathcal{E}^d}(k_p^d, p) = Enc_\mathcal{E}(k_{p,d}, p)$.
- Para un $c \in \mathcal{C}$ texto cifrado bajo $k_{p,0}$, $Dec_{\mathcal{E}^d}(k_s^d, c) = Dec_\mathcal{E}(k_{s,0}, c)$.
- El algoritmo de evaluación $Eval_{\mathcal{E}^\delta}$, toma como parámetros la clave pública k_p^δ, un circuito de profundidad menor o igual a δ con puertas en $\mathcal{P}$ y una tupla de textos cifrados por k_p^δ, y procede de la siguiente manera:

1. Añade al circuito, como paso previo entre las entradas y la primera puerta, $Dec_{\mathcal{E}}$ y convierte cada texto cifrado de entrada c_j en la tupla $(Enc_{\mathcal{E}}(k_{p,\delta-1}, k_{s,\delta j}), Enc_{\mathcal{E}}(k_{p,\delta-1}, c_j)$.
2. Se calcula $Dec_{\mathcal{E}}$ con la tupla mencionada obteniéndose los valores de entrada para el circuito original de profundidad δ y se evalúa, como en el esquema $\mathcal{E}$ las puertas a nivel δ. Los resultados parciales son devueltos junto al circuito de profundidad $\delta - 1$ que no se ha evaluado.
3. Si el circuito restante es de profundidad 0, es decir, ya hemos obtenido la salida del circuito original, se devuelve este resultado. Si no se vuelve al primer paso del algoritmo.

De esta manera, partiendo de un esquema algo homomórfico se es capaz de evaluar cualquier circuito de profundidad d elegida.

Además, es posible considerar un único par de claves k_s y k_p si el esquema original $\mathcal{E}$ es seguro circularmente o KDM seguro, es decir, se puede tomar $Enc_{\mathcal{E}}(k_p, k_s)$ sin revelar k_s y realizar el proceso anterior sin tomar la cadena de claves encriptadas descrita.

No obstante, este proceso es costoso computacionalmente, pues la evaluación homomórfica de un circuito requiere de múltiples cifrados y descifrador por cada nivel de este.

2.3.2. Esquema de Gentry

En la segunda parte de [14] Gentry describe un esquema algo homomórfico, que tras reducir el circuito de descifrado de este, el esquema llega a ser autoevaluado, y se detalla la construcción de un esquema completamente homomórfico siguiendo los pasos que se han visto.

Como se verá, este esquema está basado en problemas con retículos, similar al esquema GGH discutido en 2.2.4, y el problema de la suma de subconjuntos dispersos. Se seguirá para su descripción las variaciones sobre el esquema original publicadas en [15] que logran una implementación con tiempo de ejecución y consumo de recursos viable.

El esquema algo homomórfico de Gentry, junto a las variaciones de S. Halevi [15] y algunas de las propuestas hechas en [16] consiste en, partiendo de un retículo ideal y tomando una base privada y otra pública, menos ortogonal, considerar un vector aleatorio, y añadir a este el texto plano. De esta forma, cifrar y descifrar este texto consistirá en representar el error de este vector con respecto a un elemento del retículo mediante una base u otra.

Se verán a continuación los algoritmos de generación de claves, cifrado y descifrado que componen el esquema.

Generación de claves

1. Se considera un retículo ideal generado por la base rotacional V inducida por $I = (\vec{v})$ con $\vec{v}$ los coeficientes de un $v(x)$ en el anillo $\mathbb{Z}[x]/(x^n + 1)$, donde n es una potencia de 2.
   Debe comprobarse que la forma normal de Hermite de la base rotacional $FNH(V)$ tiene la siguiente forma:

$$FNH(V) = \begin{bmatrix} d & -r & -[r^2]_d & \cdots & -[r^{n-1}]_d \\ 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & & & \ddots & \\ 0 & 0 & 0 & \cdots & 1 \end{bmatrix} \tag{2.9}$$

   Donde $d = det(V)$ y r es una raíz de $x^n + 1$
2. Se calcula W la base rotacional asociada a $w(x)$ donde, $w(x) \cdot v(x) \equiv d$ mód $x^n + 1$.
3. Obtenemos la clave privada $k_s = \{\vec{v}, \vec{w}\}$ y la clave pública $k_p = FNH(V) = B$ que podemos representar como el par $\{d, r\}$ por la elección de $v(x)$ hecha.

Además, también se prueba en [15] que basta obtener un $v(x)$ tal que $(-r, 1, 0, \ldots, 0) \in \Delta(V)$, para que $FNH(V)$ tenga la forma 2.9.

Cifrado

1. Para cifrar un $b \in \mathcal{P} = \{0, 1\}$, se toma un vector aleatorio $\vec{u}$ tal que $u_i = 0$ con probabilidad q y $u_i = \pm 1$ con probabilidad $(1 - q)/2$.
2. Se considera $\vec{a} = 2\vec{u} + (b, 0, \ldots, 0)$.
3. Se calcula el texto cifrado $\vec{c} = \vec{a} \mod B = \vec{a} - B \cdot \lfloor B^{-1} \cdot \vec{a} \rceil$.

Al desarrollar $\vec{c}$ se señala que es posible representarlo por $c_0 = [a(r)]_d$ donde $a(x)$ es el polinomio cuyos coeficientes vienen dados por el vector $\vec{a}$, facilitando así la encriptación.

Como se ve, muchas de las decisiones tomadas en esta implementación del esquema de Gentry van orientadas a reducir el número de operaciones que se deben de realizar.

Descifrado

1. Dado un texto cifrado $\vec{c}$ se podrá obtener b calculando $\vec{a} = \vec{c} \mod V = \vec{c} - V \cdot \lfloor V^{-1} \cdot \vec{c} \rceil$.
2. Se obtiene $b = a_0 \mod 2$.

De forma similar, desarrollando el descifrado de $\vec{a}$, en la publicación original, se detalla como es posible calcular el texto plano b como $[c_0 w_i]_d \equiv b w_i \mod 2$ $i \in [0, n-1]$, si además consideramos un w_i par se tiene $b = [c_0 w_i]_d$.

Evaluación de la suma y el producto

Para la evaluación en el espacio de texto cifrado de la suma y el producto de dos textos cifrados $\vec{c}_1$ y $\vec{c}_2$, se consideran las siguientes funciones.

$$Suma_{\mathcal{E}}(\vec{c}_1, \vec{c}_2) = [\vec{c}_1 + \vec{c}_2]_d \tag{2.10}$$

$$Producto_{\mathcal{E}}(\vec{c}_1, \vec{c}_2) = [\vec{c}_1 \cdot \vec{c}_2]_d \tag{2.11}$$

Por último, para lograr un esquema completamente homomórfico se describe en la segunda parte de [15] una implementación del circuito de descifrado que, utilizando una versión de k_s ofuscada que será compartida, permite evaluar el circuito de descifrado homomórficamente y reducir el error en un texto cifrado, como se ha descrito en 2.3.1.

2.3.3. Otros esquemas completamente homomórficos

Los esfuerzos realizados en los años posteriores a la publicación del trabajo de Gentry para establecer esquemas completamente homomórficos consistieron en establecer un esquema algo homomórfico, basando parte de ellos su seguridad en los problemas en retículos ideales vistos, y utilizando el mecanismo de recifrado que se han presentado en 2.3.1 para lograr esto [17].

Al partir de esquemas algo homomórficos, la innovación de los esquemas publicados posteriormente fue principalmente la de reducir el error acumulado al evaluar circuitos honoríficamente, aumentando la profundidad que podían tener estos; o desarrollar esquemas KDM-seguros que permitieron evitar utilizar una cadena de claves para el recifrado.

Las especificaciones de algunos de estos esquemas las podemos encontrar resumidas en [17]. Como ejemplo se tiene el esquema BGV (Brakerski, Gentry y Vaikuntanathan) que parte del problema RLWE para proponer un esquema homomórfico nivelado autoevaluable, o el esquema BFV (Fan y Vercauteren basado en un criptosistema presentado por Brakerski) con muchas similitudes al esquema BGV ([18] Apéndice A y B), que se presentará en el próximo capítulo.

Otros esquemas basados en NTRU, un popular criptosistema de código abierto también basado en retículos ideales, fueron propuestos, no obstante, al igual que los demás esquemas basados en el recifrado, el coste computacional de evaluar homomórficamente el descifrado era alto.

En 2017, Cheon J., Kim, A., Kim, M. y Song, Y. presentan el esquema CKKS [19], capaz de evaluar circuitos homomórficamente utilizando aproximaciones aritméticas y reduciendo el error generado mediante un proceso que denominan reescalada del texto cifrado, permitiendo obtener aproximaciones del texto cifrado original.

El enfoque del esquema CKKS contrasta con los esquemas completamente homomórficos que se habían presentado hasta la fecha, pues aunque estos tenían un alto coste computacional, proveían resultados exactos, al contrario que la propuesta de CKKS comprometiendo la exactitud de la aritmética empleada para optimizar los cómputos.

3

Esquema BFV

En este capítulo se presentará el esquema de encriptación BFV, también denominado FV, publicado en [20] por J. Fan y F. Vercauterense y basado en un criptosistema publicado por Bakerski convirtiendo el problema LWE que subyace en este en el problema de decisión RLWE. Además, se propondrá una implementación de este esquema en el lenguaje de programación Python.

Como se verá en la siguiente sección, el esquema BFV toma el problema RLWE 2.2.3 como primitiva criptográfica con el fin de garantizar la seguridad del esquema mediante el supuesto de complejidad computacional asociado a este.

3.1. Descripción del esquema

El esquema que describen Fan y Vercauteren es, bajo una elección de parámetros adecuada, completamente homomórfico. Para ello, los autores utilizan la construcción vista en 2.3.1, pero al contrario que en el esquema de Gentry 2.3.2 el circuito de descifrado será lo suficientemente reducido para no hacer falta compactarlo, de esta manera podremos evaluar directamente el circuito de descifrado homomórficamente. Además, al ser el esquema KDM-seguro, no será necesario tomar una cadena de claves de encriptación y bastará con cifrar la clave privada.

Veamos en que consiste este esquema, presentando los algoritmos que lo componen, siguiendo para ello la notación para los problemas de RLWE vistos en 2.4.

Generación de claves

La generación de la clave privada sk consistirá en tomar una muestra de $U(\mathcal{A}_2)$.

Clave privada

1. Se elige el parámetro $k \in \mathbb{N}$ y se toma el polinomio ciclotómico $f(x) = \Phi_{2^{k+1}}(x) = x^{2^k} + 1$, de grado $d = 2^k$.

2. Se toma una muestra $s(x) \longleftarrow U(\mathcal{A}_2)$.
3. Consideramos $sk = s(x)$ como la clave privada.

Para el cálculo de la clave pública, nos centraremos en el caso en el que $\mathcal{A}_q$, el espacio del texto cifrado $\mathcal{C}$, para $q = 2^n$ con $n \in \mathbb{N}$ que nos permitirá simplificar los cálculos. Además, serán necesarias otras consideraciones con respecto a los parámetros del esquema para lograr que este sea autoevaluable, los cuales se verán más adelante.

Clave pública

1. Se elige $n \in \mathbb{N}$.
2. Se toma una muestra $a(x) \longleftarrow U(\mathcal{A}_q)$ y $e(x) \longleftarrow \chi$ donde χ es una distribución acotada por B en $\mathcal{A}$.
3. Se considera $pk = ([-(a(x) \cdot sk + e(x))]_q, a(x)) \in \mathcal{A}_q \times \mathcal{A}_q$ como la clave de cifrado.

Además de la clave de cifrado deberemos incluir en la clave pública una copia cifrada de la clave secreta, la cual nos permitirá como se vio en 2.3.1 cifrar nuevamente un texto cifrado y reducir el error contenido en este, permitiendo así evaluar homomórficamente circuitos con una profundidad deseada.

4. Calculamos $bsk = Enc_{\mathcal{E}}(pk, sk)$ siguiendo el algoritmo de cifrado que veremos en el siguiente apartado, bsk será la clave de recifrado.

Por último, como se discutirá en 3.1.2 será necesaria relinealizar el texto cifrado al evaluar un producto homomórficamente, para ello deberemos de añadir a la clave pública, una clave de relinealización rlk la cual, dependiendo del algoritmo de relinealización considerado se puede obtener como:

5. **1.º Método**
   a) Se considera $T \in \mathbb{N}$.
   b) Muestreamos para $i \in \{0, \ldots, \lfloor \log_T(q) \rfloor\}$, $a_i(x)$ $U(\mathcal{A}_q)$ y $e_i(x) \longleftarrow \chi$.
   c) Se obtiene:

$$rlk = \{([-(a_i(x) \cdot sk + e_i(x)) + T^i \cdot sk^2]_q, a_i(x)) : i \in \{0, \ldots, \lfloor \log_T(q) \rfloor\}\}. \tag{3.1}$$

   2.º Método:
   a) Se considera $p \in \mathbb{N}$.
   b) Muestreamos para $a(x) \longleftarrow U(\mathcal{A}_{pq})$ y $e(x) \longleftarrow \chi'$, donde χ' es una distribución acotada por B' en $\mathcal{A}$.
   c) Se obtiene:

$$rlk = ([-(a(x) \cdot sk + e(x)) + p \cdot sk^2]_{pq}, a(x)). \tag{3.2}$$

6. Se considera (pk, bsk, rlk) la tupla de claves que se deberá hace pública para poder trabajas con el esquema completamente homomórfico.

No obstante, bastará pk para poder cifrar un texto, y (pk,rlk) para lograr un esquema algo homomórfico.

Cifrado

Denotaremos por $\triangle$ a $\lfloor \frac{q}{t} \rfloor$, donde $\lfloor \cdot \rfloor$ es el redondeo al menor entero. Si consideramos $t|q$ donde $t = 2^m \in \mathbb{N}$ como el módulo de $\mathcal{A}_t$ el espacio del texto plano $\mathcal{P}$, se tiene que $\triangle = \frac{q}{t}$.

1. Codificamos el mensaje m en el texto plano $m(x) \in \mathcal{A}_t$.
2. Se toma $u(x) \longleftarrow U(\mathcal{A}_2)$ y $e_1(x), e_2(x) \longleftarrow \chi$.
3. Se calcula el texto cifrado

$$c = ([pk_1(x) \cdot u(x) + e_1(x) + \triangle \cdot m(x)]_q, [pk_2(x) \cdot u(x) + e_2(x)]_q). \tag{3.3}$$

Descifrado

El algoritmo de descifrado es el siguiente:

1. Sea el texto cifrado $c = (c_0(x), c_1(x))$.
2. Se calcula el texto plano

$$m(x) = \left[\left\lfloor \frac{t \cdot [c_0(x) + c_1(x) \cdot sk]_q}{q} \right\rceil \right]_t \tag{3.4}$$

En el caso particular en el que $q = 2^n$ y $t|q$ podemos obtener $m(x)$ como:

$$m(x) = \left[\left\lfloor \frac{c_0(x) + c_1(x) \cdot sk}{2^{-(m-n)}} \right\rceil \right]_t, \text{para } t = 2^n. \tag{3.5}$$

Veamos que el algoritmo de descifrado descrito permite obtener $m(x)$ correctamente. Se tiene que:

$$[c_0(x) + c_1(x) \cdot sk]_q = [pk_1(x) \cdot u(x) + e_1(x) + \triangle \cdot m(x) + (pk_2(x) \cdot u(x) + e_2(x)) \cdot sk]_q.$$

Sustituyendo $pk_i(x)$,

$$\begin{aligned} [c_0(x) + c_1(x) \cdot sk]_q = [-(a(x) \cdot sk + e(x)) \cdot u(x) + e_1(x) + \triangle \cdot m(x) + \dots \\ \dots + (a(x) \cdot u(x) + e_2(x)) \cdot sk]_q = \\ = [\triangle \cdot m(x) - e(x) \cdot u(x) + e_1(x) + e_2(x) \cdot sk]_q. \end{aligned}$$

Se puede ver que, $\triangle \cdot m(x) \in \mathcal{A}_q$ no obstante, el ruido en el texto cifrado $v = -e(x) \cdot u(x) + e_1(x) + e_2(x) \cdot sk \in \mathcal{A}$. Si consideramos:

$$c_0(x) + c_1(x) \cdot sk = \triangle \cdot m(x) + v + q \cdot r(x), \text{para cierto } r(x) \in \mathcal{A}, \tag{3.6}$$

y multiplicamos por t/q:

$$\frac{t}{q}(c_0(x) + c_1(x) \cdot sk) = m(x) + \frac{t}{q} \cdot (v - (q/t - \triangle)m(x)) + t \cdot r(x).$$

Se tiene que, como $q \cdot r(x) \equiv 0 \mod q$:

$$\|\frac{t}{q}(v - (q/t - \triangle)m(x))\| < \frac{1}{2},$$

Donde $\|\cdot\|$ denota la norma L_∞ sobre el vector de coeficientes del polinomio, puesto que es de interés acotar el ruido en cada coeficiente.

Teniendo en cuenta que $m(x) \in \mathcal{A}_t$, se tiene:

$$\|v\| < \frac{\triangle}{2}.$$

Al ser $e(x), e_1(x), e_2(x)$ muestras de χ y $u(x)$ y sk elementos de $\mathcal{A}_2$ se tiene que $\|v\| < 3B$. Por tanto, el descifrado será correcto si $B < \frac{\triangle}{6}$, y en el caso $q = 2^n$ y $t|q$ se tiene que $B < \frac{2^{n-m-1}}{3}$.

Se puede ver en 3.1 que al suponer $m(x) = 3$ el descifrado es correcto si $3\triangle + v \in ((3 - 1/2)\triangle, (3 + 1/2)\triangle]$,puesto que $[\triangle m(x) + v]_q = \triangle m(x)$, es decir si $\|v\| < \triangle/2$ como hemos visto. En caso contrario, el descifrado no devolverá los coeficientes de $m(x)$ correctamente.

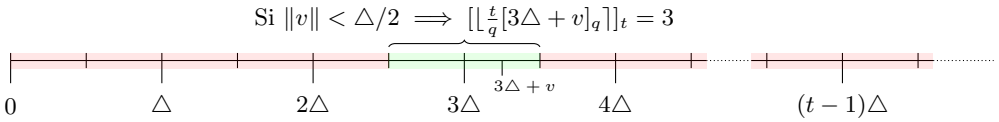


Figura 3.1: Ejemplo de intervalo en el que se produce el descifrado correctamente para $m(x) = 3$.

3.1.1. Evaluación de la suma

Para dos textos cifrados $c^{(1)} = (c_1^{(1)}(x), c_2^{(1)}(x))$, $c^{(2)} = (c_1^{(2)}(x), c_2^{(2)}(x))$ según el algoritmo que se ha descrito previamente. Se tiene que:

$$\begin{aligned}[c_1^{(1)}(x) + c_2^{(1)}(x) \cdot sk]_q + [c_1^{(2)}(x) + c_2^{(2)}(x) \cdot sk]_q = [c_1^{(1)}(x) + c_1^{(2)}(x) + \dots \\ \dots + (c_2^{(1)}(x) + c_2^2(x)) \cdot sk]_q = \triangle(m^{(1)}(x) + m^{(2)}(x)) + v^{(1)} + v^{(2)} = \\ = \triangle[m^{(1)}(x) + m^{(2)}(x)]_t + v^{(1)} + v^{(2)} - (q/t - \triangle)t \cdot r(x).\end{aligned}$$

Ya que $m^{(1)}(x) + m^{(2)(2)}(x) = \left[m^{(1)}(x) + m^{(2)}(x)\right]_t + t \cdot r(x)$. El nuevo ruido que tendrá la suma de los textos cifrados será $v_{suma} = v^{(1)} + v^{(2)} - (q/t - \triangle)t \cdot r(x)$ y estará acotado por $\|v_{suma}\| < 2(3B) + t$.

Por tanto, podemos calcular la suma de dos textos cifrados como:

$$Suma_{\mathcal{E}}(c^{(1)}, c^{(2)}) = ([(c_1^{(1)}(x) + c_1^{(2)}(x)]_q, [c_2^{(1)}(x) + c_2^{(2)}(x)]_q). \tag{3.7}$$

3.1.2. Evaluación del producto

Para evaluar el producto de dos textos cifrados requerimos realizar dos pasos. Primero, como veremos a continuación, deberemos calcular el producto de los textos cifrado y escalarlo, tras esto deberemos relinealizar el resultado mediante uno de los métodos que serán descritos.

Veamos como obtener el producto de dos textos cifrados c^1, c^2. Considerando la igualdad 3.6 se tiene:

$$\begin{aligned}(c_1^{(1)}(x) + c_2^{(1)}(x) \cdot sk) \cdot (c_1^{(2)}(x) + c_2^{(2)}(x) \cdot sk) = \triangle^2 m^{(1)}(x) m^{(2)}(x) + \\ \triangle \cdot (m^{(1)}(x) v^{(2)} + m^{(2)}(x) v^{(1)}) + v^{(1)} v^{(2)} + q \cdot (v^{(1)} r^{(2)} + v^{(2)} r^{(1)}) + \\ q\triangle \cdot (m^{(1)}(x) r^{(2)} + m^{(2)}(x) r^{(1)}) + q^2 r^{(1)} r^{(2)}\end{aligned} \tag{3.8}$$

para $r^{(1)}, r^{(2)} \in \mathcal{A}$.

Multiplicando t/q, y escribiendo $m^{(1)(x)+m^{(2)}(x)} = [m^{(1)(x)+m^{(2)}(x)}]_t + t \cdot r_m$ y,$v^{(1)} + m^{(2)} = [v^{(1)} + m^{(2)}]_\triangle + \triangle \cdot r_v$.

Y tomando la aproximación:

$$\begin{aligned}\frac{t}{q} \cdot [(c_1^{(1)}(x) + c_2^{(1)}(x) \cdot sk) \cdot (c_1^{(2)}(x) + c_2^{(2)}(x) \cdot sk)] = \lfloor t/q \cdot c_o(x) \rceil + \\ \lfloor t/q \cdot c_o(x) \rceil \cdot sk + \lfloor t/q \cdot c_o(x) \rceil \cdot sk^2 + r_a\end{aligned} \tag{3.9}$$

con $r_a \in \mathcal{A}$.

Se concluye en el *Lema 2* de [20] que:

$$\begin{aligned}[[(c_1^{(1)}(x) + c_2^{(1)}(x) \cdot sk) \cdot (c_1^{(2)}(x) + c_2^{(2)}(x) \cdot sk)] = \lfloor t/q \cdot c_o(x) \rceil + \lfloor t/q \cdot \\ c_o(x) \rceil \cdot sk + \lfloor t/q \cdot c_o(x) \rceil \cdot sk^2]_q = \triangle [m^1 m^2] + v_3\end{aligned} \tag{3.10}$$

con $\|v_3\| < 2 \cdot \delta_{\mathcal{A}} \cdot t \cdot E \cdot (\delta_{\mathcal{A}} \|sk\| + 1) + 2t^2 \delta_{\mathcal{A}}^2 \cdot (\|sk\| + 1)^2$, donde $\delta_{\mathcal{A}} = \text{máx}(\|ab\|/(\|a\|\|b\|) : a, b \in \mathcal{A})$ se denomina el factor de expansión de $\mathcal{A}$ acotado por $\sqrt{d}$, $\|v\| \leq E < \triangle/e$, y como $sk \in U(\mathcal{A}_2)$ se tiene que $\|sk\| = 1$.

Por tanto, se tiene que el producto de dos textos cifrados c^1, c^2 es:

$$Producto_{\mathcal{E}}(\lfloor c^{(1)}, c^{(2)}) = ([\lfloor \frac{t}{q} \cdot c_1^{(1)}(x) \cdot c_1^{(2)}(x) \rceil]_q, [\lfloor \frac{t}{q} \cdot c_1^{(1)}(x) \cdot c_1^{(2)}(x) + \\ + c_2^{(1)}(x) \cdot c_2^{(2)}(x) \rceil]_q, [\lfloor \frac{t}{q} \cdot c_2^{(1)}(x) \cdot c_2^{(2)}(x) \rceil]_q). \tag{3.11}$$

Como se ve en 3.11 el producto de dos textos cifrados devuelve un elemento de $\mathcal{A}_q \times \mathcal{A}_q \times \mathcal{A}_q$ sin embargo, los textos cifrados mediante el esquema están formados por una dupla. Con el fin de obtener una dupla que represente al producto, se describe en [20] dos maneras de relinealizarlo de forma que

$$[c_0(x) + c_1(x)sk + c_x(x)sk^2]_q = [c_0'(x) + c_1'(x)sk + r(x)]_q, \tag{3.12}$$

para cierto $r(x)$ cuya norma sea negligible.

1.º Método de relinalización

La primera forma descrita consiste en escribir $c_2(x)$ en una nueva base T independiente de t y q, de forma que $c_2(x) = [\sum_{i=0}^{l} T^i c_2^{(i)}]_q$ con $l = \lfloor \log_T(q) \rfloor$.

De esta manera, si se toma la clave de relinalización rlk según se ha definido en 3.1 y se considera:

$$c_0'(x) = [c_0(x) + \sum_{i=0}^{l} rlk_0^{(i)} c_2^{(i)}(x)]_q, c_1'(x) = [c_1(x) + \sum_{i=0}^{l} rlk_1^{(i)} c_2^{(i)}(x)]_q, \tag{3.13}$$

se tiene que:

$$[c_0'(x) + c_1'(x)sk]_q = [c_0(x) + c_1(x)sk + c_2(x)sk^2 - \sum_{i=0}^{l} c_2^{(i)}(x)e_i(x) \mod q]_q. \tag{3.14}$$

Podemos considerar entonces $r(x) = -\sum_{i=0}^{l} c_2^{(i)}(x)e_i(x) \mod q$ en 3.12 como el error generado al relinealizar el texto cifrado acotado por, $(l+1)BT\delta_{\mathcal{A}}/2$.

2.º Método de relinalización

Otra forma propuesta para obtener una dupla a partir del producto de los textos cifrados consiste en expresar sk^2 codificado módulo $p \cdot q$ para un entero p como se describe en 3.2.

De esta manera podemos tomar:

$$c_0'(x) = [c_0(x) + \lfloor rlk_0 c_2(x) \rceil]_q, c_1'(x) = [c_1(x) + \lfloor rlk_1 c_2(x) \rceil]_q \tag{3.15}$$

como la nueva dupla c' que permite representar el producto con un error $\|r(x)\| < \frac{qB'\delta_{\mathcal{A}}}{p} + \frac{\delta_{\mathcal{A}}\|sk\|+1}{2}$.

3.2. Implementación del esqeuma

Para la implementación del esquema se ha elegido el lenguaje de programación Python [21] en su versión 3.10.6 debido a su virtud para el desarrollo rápido de programas, y la facilidad de lectura del código escrito en este, los cuales nos permiten presentar la siguiente implementación del esquema de una forma más didáctica y accesible en comparación a otros lenguajes.

Se usará la librería numpy [22] en su versión 1.23.2 la cual incluye una gran variedad clases y funciones que facilitan operaciones algebraicas comunes. En particular se considera esta librería, ya que incluye funciones para operar con polinomios y para el muestreo de una distribución normal.

Veamos a continuación el código empleado, comenzando por la importación de la librería mencionada.

```
1 import numpy as np
2 from numpy.polynomial import polynomial as pol
```

Definimos las funciones `sumpol` y `mulpol`, las cuales aprovechando la implementación de las operaciones con polinomios en $\mathbb{R}[x]$ de la librería numpy, permiten operar con polinomios en $\mathcal{A} = \mathbb{Z}[x]/(f(x))$ para $f(x) \in \mathbb{Z}[x]$. En particular las usaremos para operar en este anillo, restringiendo los coeficientes a $\mathbb{Z}_q : q \in \mathbb{Z}$, es decir en $\mathcal{A}_q$ como se ha visto en 3.1.

Para representar los polinomios usaremos objetos `ndarrays` de la librería numpy. Estos objetos representan un conjunto ordenado elementos al igual que una lista, no obstante permiten un mejor rendimiento. Además, las listas son convertidas a objetos de este tipo por las funciones de numpy. Los coeficientes de los polinomios se almacenan de menor a mayor grado.

Estas funciones permiten elegir el módulo que se les aplica a los coeficientes `modint` y el módulo que se aplica al polinomio resultante `modpol`, además permiten realizas sumatorios y productorios.

```
4  def sumpol(*args, modint=None, modpol=None):
5      """Sums multiple polynomials on the ring Z_{modint}[x]/(modpol)
6      Parameters:
7          *args (list[int]): polynomials in Z[x] passed each one as a list of its
   ↪  coefficients from lesser to greater.
8          modint (Optional[int]): integer indicating the set of the coefficients
   ↪  Z_{modint}, defaults to None.
9          modpol (Optional[list[int]]): polynomial indicating the quotient ring
   ↪  Z[x]/(modpol), defaults to None.
10     Returns:
11         array[np.int64]: the cumulative sum of the polynomials in *args as an
   ↪  element of Z_{modint}[x]/(modpol) if such parameters are set.
12     """
13     val = [0]
```

```
    for poly in args:
        val = pol.polyadd(val, poly)
    if modint is not None:
        val %= modint
    if modpol is not None:
        val = pol.polydiv(val, modpol)[1]
        if modint is not None:
            val %= modint

    return np.rint(val).astype(np.int64)

def mulpol(*args, modint=None, modpol=None):
    """Multiplies multiple polynomials on the ring Z_{modint}[x]/(modpol)
    Parameters:
        *args (list[int]): polynomials in Z[x] passed each one as a list of its coefficients from lesser to greater.
        modint (Optional[int]): integer indicating the set of the coefficients Z_{modint}, defaults to None.
        modpol (Optional[list[int]]): polynomial indicating the quotient ring Z[x]/(modpol), defaults to None.
    Returns:
        array[np.int64]: the cumulative product of the polynomials in *args as an element of Z_{modint}[x]/(modpol) if such parameters are set.
    """
    val = [1]
    for poly in args:
        val = pol.polymul(val, poly)
    if modint is not None:
        val %= modint
    if modpol is not None:
        val = pol.polydiv(val, modpol)[1]
        if modint is not None:
            val %= modint

    return np.rint(val).astype(np.int64)
```

Definimos las funciones `get_unifpol` y `get_normpol` que nos permiten tomar una muestra de $U(\mathcal{A}_{modint})$ y χ una distribución normal entera en $\mathcal{A}$.

```
def get_unifpol(degree, modint=2):
    """Generates a polynomial of the specified degree with coefficients sampled from a uniform distribution in Z_{modint}.
    Parameters:
        degree (int): degree of the polynomial.
        modint (int): modulo for the coefficients.
    Returns:
        array[np.int64]: polynomial sampled from  U(Z_{modint}[x])
     """
    return np.random.randint(0, modint, size=degree+1,dtype=np.int64)

def get_normpol(degree, mean=0, var=1):
    """Generates a polynomial of the specified degree with coefficients  sampled from a normal distribution in Z.
```

```
58      Parameters:
59          degree (int): degree of the polynomial.
60          mean (float): mean of the normal distribution.
61          var (float): variance of the normal distribution.
62      Returns:
63          array[np.int64]: polynomial with integer coefficients sampled from
    ↪ N(mean, var)
64      """
65      return np.rint(np.random.normal(mean, np.sqrt(var), size=degree+1))
```

A continuación se presenta las principales funciones que implementan el esquema, las cuales se basan en los algoritmos presentados en la sección anterior.

Para facilitar su lectura se han tomado nombres de variables que coinciden con los utilizados en la sección anterior.

Generación de clave pública y privada

Para la generación de la clave pública y privada elegimos los parámetros, `q` para el módulo de los coeficientes, `fx` el polinomio ciclotómico de grado 2^k, $x^{2^k} + 1$ y `var` la varianza considerada para la distribución χ que se ha definido previamente.

```
67  def keygen(q, fx, var):
68      '''Generates the public and private key pair for the BFV public key
69      encryption cryptosystem.
70      Parameters:
71          q (int): modulo for the coefficients of the polynomials.
72          fx (list[int]): polynomial for the quotient ring Z[x]/(fx).
73          var (float): variance for the error sampling.
74      Returns:
75          tuple[tuple[array[np.int64]], array[mp.int64]]: returns the public and
    ↪ private key pair.
76      '''
77      n = len(fx)-1
78      s = get_unifpol(n-1, 2)
79      a = get_unifpol(n-1, q)
80      e = get_normpol(n-1, 0, var)
81      b = sumpol(mulpol(-a, s, modint=q, modpol=fx), -e, modint=q, modpol=fx)
82
83      return ((b, a), s)
```

Como se ve, `keygen` toma sendas muestras de los polinomios `s`, `a` y `e` y calcula la clave pública, devolviendo esta y la clave privada en una tupla.

A continuación se presenta a modo de ejemplo el uso de esta y el resto de funciones que implementan el esquema a través del intérprete de Python, tras haber importado todo el código que se muestra en esta sección.

Para el este ejemplo se considera $\mathcal{A} = \mathbb{Z}[x]/(x^4+1)$ y coeficientes en $\mathbb{Z}_{2^{14}}$ para el texto cifrado, es decir en $\mathcal{A}_{2^{14}}$. Utilizando la función `keygen` se obtienen la clave pública `pk` y privada `sk`:

```
>>> q = 2**14
>>> fx = [1, 0, 0, 0, 1]
>>> var = 2
>>> pk, sk = keygen(q, fx, var)
>>> pk, sk
((array([ 8254,  5777, 15745,  9057], dtype=int64), array([14278,  2745,  4583,
↪  3670], dtype=int64)), array([0, 1, 1, 0], dtype=int64))
```

Como se ve la clave pública $\texttt{pk} = (8254+5777x+15745x^2+9057x^3, 14278+2745x+4583x^2+3670x^3)$ y la clave privada $\texttt{sk} = x + x^2$ se encuentran en $\mathcal{A}_{2^{14}} \times \mathcal{A}_{2^{14}}$ y $\mathcal{A}_2$ respectivamente.

Cifrado

Para la función de cifrado se considera un texto plano $\texttt{m} \in \mathcal{A}_t$ que se pasará a la función `encrypt` como un array de `numpy` y se calcula como se ha visto $c = (\texttt{c0}, \texttt{c1})$.

```
85  def encrypt(pk, m, t, q, fx, var):
86      '''Encrypts a plaintext using the BFV cryptosystem.
87      Parameters:
88          pk (tuple[array[int]]): public key as a tuple of two polynomials.
89          m (array[int]): plaintext encoded in a polynomial of Z_{t}[x]/(fx)
90          t (int): modulo for the coefficients of the plaintext polynomials.
91          q (int): modulo for the coefficients of the ciphertext polynomials.
92          fx (list[int]): polynomial for the quotient ring Z[x]/(fx)
93          var (float): variance for the error sampling.
94      Returns:
95          tuple[array[np.int64]]: returns a tuple representing the ciphertext of
    ↪  n.
96      '''
97      n = len(fx)-1
98      e1 = get_normpol(n-1, 0, var)
99      e2 = get_normpol(n-1, 0, var)
100     u = get_unifpol(n-1)
101     c0 = sumpol(
102         mulpol(pk[0], u, modint=q, modpol=fx),
103         e1, q // t * m, modint=q, modpol=fx)
104     c1 = sumpol(
105         mulpol(pk[1], u, modint=q, modpol=fx),
106         e2, modint=q, modpol=fx)
107
108     return (c0, c1)
```

Siguiendo con el ejemplo anterior, considerando $t = 8$ y el texto plano $\texttt{m1} = 2 \in \mathcal{A}_8$.

```
>>> t = 8
>>> m1 = np.array([2])
>>> ct1 = encrypt(pk, m1, t, q, fx, var)
>>> ct1
(array([14703,   640,  7327,  8255], dtype=int64), array([13642, 11803, 12714,
↪  14278], dtype=int64)
```

Se obtiene el texto cifrado `ct1` como una tupla (`c0`, `c1`) formada por los polinomios con los coeficientes que hemos obtenido.

Descifrado

La función `decrypt` toma un texto cifrado `c` y junto a la clave privada `sk` y devuelve el texto plano original `dec_pol`.

```
110 def decrypt(sk, c, t, q, fx):
111     '''Decrypts a ciphertext using the BFV cryptosystem.
112     Parameters:
113         sk (array[int]): secret key.
114         c (array[int]): ciphertext to be decrypted.
115         t (int): modulo for the coefficients of the plaintext polynomials.
116         q (int): modulo for the coefficients of the ciphertext polynomials.
117         fx (list[int]): polynomial for the quotient ring Z[x]/(fx)
118     Returns:
119         array[np.int64]: returns an array with the coefficients of the decrypted
    ↪  polynomial of c.
120     '''
121     dec_pol = np.rint(t * sumpol(
122         mulpol(c[1], sk, modint=q, modpol=fx),
123         c[0],  modint=q, modpol=fx) / q) % t
124 
125     return dec_pol
```

Para el texto que hemos cifrado previamente `m1` se tiene:

```
>>> d1 = decrypt(sk, ct1, t, q, fx)
>>> d1
array([2., 0., 0., 0.])
```

Obtenemos `d1` = `m1` = 2, el texto plano original.

Suma en el espacio de texto cifrado

Para la suma de textos cifrados se ha definido la función `sum_ct` la cual devuelve el sumatorio homomórfico `sum` de los textos cifrados `ct[i]` como un elemento de $\mathcal{A}_q \times \mathcal{A}_q$ que como hemos visto, bajo la elección adecuada de parámetros para el esquema, es equivalente al cifrado de suma de los `m[i]` textos planos asociados en $\mathcal{A}_t$.

En la práctica, debido a que el error que se añade al texto cifrado tras cada suma no tiene una magnitud considerable, se pueden evaluar circuitos formados solo por esta función un número elevado de veces, en comparación con el producto.

```
127 def sum_ct(*ct, q, fx):
128     '''Sums multiple ciphertexts encrypted using the BFV cryptosystem.
129     Parameters:
130         *ct (list[array[int]]): list of ciphertexts.
131         q (int): modulo for the coefficients of the ciphertext polynomials.
132         fx (list[int]): polynomial for the quotient ring Z[x]/(fx)
133     Returns:
134         list(array[np.int64]): ciphertext of the sum.
135     '''
136     sum = [[0], [0]]
137     for c in ct:
138         sum = [sumpol(sum[0], c[0], modint=q, modpol=fx),
139             sumpol(sum[1], c[1], modint=q, modpol=fx)]
140 
141     return sum
```

Por ejemplo, tomando $\texttt{m2} = 4$ y $\texttt{m3} = 5$ podemos evaluar la suma del cifrado de estos y `ct1` homomórficamente.

```
>>> m2, m3 = np.array([4]), np.array([5])
>>> ct2, ct3 = encrypt(pk, m2, t, q, fx, var), encrypt(pk, m3, t, q, fx, var)
>>> ct_s = sum_ct(ct1, ct2, ct3, q=q, fx=fx)
>>> ct_s
[array([ 5671, 14221, 10859, 11968], dtype=int64), array([ 9594, 12323,  8488,
↪  7421], dtype=int64)]
```

Y tras descifrar `ct_s` se obtiene `d_s`.

```
>>> d_s = decrypt(sk, ct_s, t, q, fx)
>>> d_s
array([3., 0., 0., 0.])
```

Por un lado, se tiene $\texttt{d2} = 3$ y, por otra parte, se puede ver que $\texttt{m1} +_{\mathcal{A}_t} \texttt{m2} +_{\mathcal{A}_t} \texttt{m3} = [2 + 4 + 5]_8 = [3]_8 = \texttt{d2}$. Vemos de esta forma como es posible evaluar la suma de textos cifrados homomórficamente.

Producto en el espacio de texto cifrado

La evaluación homomórfica del producto de dos textos cifrados conlleva, como se ha discutido en 3.1.2, el cálculo del producto propiamente y la posterior relinealización de este mediante alguno de los métodos vistos.

Como paso previo a presentar las funciones que realizaran estas tareas, se define la función auxiliar `rep_base` la cual nos permite obtener una lista con los dígitos `digits` de un número entero `number` representado en una base

determinada `base`. Esta función será de utilidad para utilizar el 1.º método de relinealización que se ha mostrado en 3.14.

```
143 def rep_base(number, base):
144     """Returns the digits of an integer base 10 in a new base.
145     Parameters:
146         number (int): number in base 10.
147         base (int): base to represent the number.
148     Returns:
149         list[int]: digits of number in the new base from lesser to greater.
150     """
151     if number == 0:
152         return [0]
153     else:
154         digits = []
155         while number:
156             digits.append(number % base)
157             number //= base
158         return digits
```

Como ejemplo se puede ver los dígitos de 25 en base 2, $(11001)_2$.

```
>>> rep_base(25, 2)
[1, 0, 0, 1, 1]
```

Con el fin de relinealizar el producto de dos textos cifrados, se deberá establecer antes una clave a tal efecto según el método elegido. Para ello se usará la función `rl_keygen`, que implementa la generación de la clave de relinalización tanto para el 1.º método 3.1 como para el 2.º método 3.2 vistos.

Para el 1.º método se consideran los parámetros `rlmode`= 1 para la elección de este, `rlpar`= T como la base en la que se descompondrá `c2`, y `rlvar`= `var` la varianza establecida previamente.

Por otra parte, para el 2.º método la función considera `rlmode`= 2 para la elección de este, `rlpar`= p para establecer el módulo $p \cdot q$, y `rlvar` un nuevo parámetro que establecerá la varianza según se ha descrito en 3.1.2.

```
160 def rl_keygen(sk, q, fx, rlmode, rlpar, rlvar):
161     ''' Generates the relinearization key for the BFV cryptosystem.
162     Parameters:
163         sk (array[int]): secret key.
164         q (int): modulo for the coefficients of the ciphertext polynomials.
165         fx (list[int]): polynomial for the quotient ring Z[x]/(fx)
166         rlmode (int): method of relinearization, accepted values are 1 or 2.
167         rlpar (int): parameter for the chosen method.
168         rlvar (float): variance for the chosen method.
169     Returns:
170         list(array[np.int64]): relinearization key for the chosen method.
171     '''
172     n = len(fx)-1
```

```
    if rlmode == 1:
        l = np.floor(np.log(q) / np.log(rlpar)).astype(np.int64)
        rlk = []
        for i in range(l + 1):
            a = get_unifpol(n-1, q)
            e = get_normpol(n-1, 0, rlvar)
            b = np.rint(sumpol(
                mulpol(-a, sk, modpol=fx),
                sumpol(-e, (rlpar ** i) * mulpol(sk, sk),  modpol=fx), modint=q,
                ↪  modpol=fx)).astype(np.int64)
            rlk.append([b, a])
        return rlk

    elif rlmode == 2:
        rlmod = q * rlpar
        a = get_unifpol(n-1, rlmod)
        e = get_normpol(n-1, 0, rlvar)
        b = sumpol(mulpol(-a, sk, modpol=fx),
            sumpol(-e, rlpar * mulpol(sk, sk), modpol=fx), modpol=fx) % rlmod

        return b, a

    else:
        raise ValueError(f'Invalid rlmode value {rlmode}; should be "1" or "2"
        ↪  according to the relinearization method chosen.')
```

Veamos un para de ejemplos de claves de relinealización utilizando cada método.

```
>>> T = 10
>>> rlk1 = rl_keygen(sk, q, fx, 1, T, var2)
>>> rlk1
[[array([ 2680,  9915, 11196], dtype=int64), array([15362,  6212, 10174,  8892],
↪  dtype=int64)], [array([ 5605,  2746, 12027,    80], dtype=int64), array([
↪  3649,   719, 15605,  6393], dtype=int64)], [array([ 2728, 11707, 10061,
↪  4019], dtype=int64), array([ 8874, 13933, 15016,  4196], dtype=int64)],
↪  [array([ 2236, 10572, 11030,  6145], dtype=int64), array([ 1582,  4772,  7467,
↪  12154], dtype=int64)], [array([ 4196, 10772,  5644,  4455], dtype=int64),
↪  array([ 4308,    48, 15499, 15081], dtype=int64)]]
```

Obteniendo $l = \lfloor \log_T(q) \rfloor = 5$ duplas con polinomios en $\mathcal{A}_q$ que codifican $sk^2 \cdot T^i$.

Para el 2.º método consideramos $\texttt{t}{=}q^2$ y χ' con una varianza $\texttt{var2} = 2$.

```
>>> var2 = 1
>>> p = q**2
>>> rlk2 = rl_keygen(sk, q, fx, 2, p, var2)
>>> rlk2
(array([3030213365461,  688167781624, 3772907154672, 2504115316671], dtype=int64),
↪  array([ 536626872920,   88780918967, 1805687146376, 1224794654543],
↪  dtype=int64))
```

Obteniendo una dupla que codifica sk^2 con dos polinomios en $\mathcal{A}_{p \cdot q=2^{30}}$.

Por último, para la evaluación homomórfica del producto y su posterior relinalización, si así se desea, se tiene la función `mul_ct` que toma dos textos cifrados `ct1` y `ct2` y los parámetros: `rlmode` para elegir el método de relinealización, `rlk` la clave que hemos generado previamente, y `rlpar` un parámetro propio para cada método, `T` o `p` según se han denotado.

```
197 def mul_ct(ct1, ct2, t, q, fx, rlmode=None, rlk=None, rlpar=None):
198     ''' Multiplies two ciphertexts encrypted using the BFV cryptosystem.
199     Parameters:
200         ct1, ct2 (tuple[array[int]]): ciphertexts to be multiplied
201         t (int): modulo for the coefficients of the plaintext polynomials.
202         q (int): modulo for the coefficients of the ciphertext polynomials.
203         fx (list[int]): polynomial for the quotient ring Z[x]/(fx)
204         rlmode (Optional[int]): method of relinearization, accepted values are 1
    ↪  or 2.
205         rlpar (int): parameter for the chosen method.
206         rlvar (float): variance for the chosen method.
207     Returns:
208         list(array[np.int64]): relinearization key for the chosen method.
209     '''
210     #Raises an error if some of the relinearization parameters are not set.
211     if rlmode is not None and (rlk is None or rlpar is None):
212         raise ValueError(f'Value of rlk or rlpar not set; value should be set if
        ↪  rlmode is not None.')
213 
214     prod = [np.rint(mulpol(ct1[0], ct2[0], modpol=fx) * t / q) % q,
215         np.rint(sumpol(mulpol(ct1[0], ct2[1], modpol=fx), mulpol(ct1[1], ct2[0],
        ↪  modpol=fx), modpol=fx) * t / q) % q,
216         np.rint(mulpol(ct1[1], ct2[1], modpol=fx) * t / q) % q]
217 
218     if rlmode is None:
219         #If no relinearization method is chosen returns only the mon
        ↪  relinealized product
220         return prod, None
221 
222     elif rlmode==1:
223 
224         l = np.floor(np.log(q) / np.log(rlpar)).astype(np.int64)
225         coef_baseT = [rep_base(coef, rlpar) for coef in prod[2]]
226         coef_baseT = [digits + [0]*(l+1-len(digits)) for digits in coef_baseT]
227 
228         c_20, c_21 = [0], [0]
229         for i in range(l + 1):
230             c2i= [digits[i] for digits in coef_baseT]
231             c_20 = sumpol(c_20, mulpol(rlk[i][0], c2i, modpol=fx), modpol=fx)
232             c_21 = sumpol(c_21, mulpol(rlk[i][1], c2i, modpol=fx), modpol=fx)
233 
234         rlprod = [ sumpol(prod[0], np.rint(c_20) % q, modint=q, modpol=fx),
235                    sumpol(prod[1], np.rint(c_21) % q, modint=q, modpol=fx) ]
236 
237         return prod, rlprod
```

```
238
239     elif rlmode==2:
240
241         rlprod = [
242             sumpol(prod[0], np.rint(mulpol(prod[2], rlk[0], modpol=fx) / rlpar)
                ↪  % q, modpol=fx) % q,
243             sumpol(prod[1], np.rint(mulpol(prod[2], rlk[1], modpol=fx) / rlpar)
                ↪  % q, modpol=fx) % q
244         ]
245
246         return prod, rlprod
247
248     #Raises an error if the relinearization mode parameter is not 1 or 2.
249     else:
250         raise ValueError(f'Invalid rlmode value {rlmode}; should be "1" or "2"
            ↪  according to the relinearization method chosen.')
```

Obtenemos una dupla formada por el producto propiamente (c_0, c_1, c_2), y su relinealización (c'_0, c'_1).

Siguiendo el ejemplo anterior, consideramos `ct1` y `ct2` previamente usados y tememos para cada método de relinealización los siguientes resultados para el producto $\texttt{ct1} \cdot \texttt{ct2}$.

```
>>> ct_mul1 = mul_ct(ct1, ct2, t, q, fx, 1, rlk1, T)
>>> ct_mul1
([array([11589., 10678.,  3168., 15348.]), array([10650.,  5782., 15498.,
↪  16196.]), array([16030., 12073.,  5460., 15152.])],
[array([14400, 13936, 11471,  9599], dtype=int64), array([ 7372, 13809,  9459,
↪  5169], dtype=int64)])
>>> ct_mul2 = mul_ct(ct1, ct2, t, q, fx, 2, rlk2, p)
>>> ct_mul2
([array([11589., 10678.,  3168., 15348.]), array([10650.,  5782., 15498.,
↪  16196.]), array([16030., 12073.,  5460., 15152.])],
[array([ 7532,   330,  9165, 13076], dtype=int64), array([13625,  9844,  9921,
↪  14200], dtype=int64)])
```

Con ambos métodos hemos obtenido una tupla de 3 polinomios, iguales en ambos casos, que corresponde al producto sin relinealizar. Sin embargo, como se ha resaltado, la dupla correspondiente al producto relinealizado difiere en ambos casos. Para finalizar, se procede a comprobar si el producto relinalizado mediante cada método coincide con el producto de los textos planos asociados.

```
>>> d_m1 = decrypt(sk, ct_mul1[1], t, q, fx)
>>> d_m1
array([0., 0., 0., 0.])
>>> d_m2 = decrypt(sk, ct_mul2[1], t, q, fx)
>>> d_m2
array([0., 0., 0., 0.])
```

Tras descifrar el resultado obtenido usando ambos métodos se tiene $\texttt{d\_m1} = \texttt{d\_m2} = 0$ que como podemos observar coincide con el producto de `m1` y `m2` en el espacio de texto plano, $\texttt{m1} \cdot_{\mathcal{A}_\sqcup} \texttt{m2} = [2 \cdot 4]_8 = [0]_8$.

3.3. Otras implementaciones disponibles

Con el fin de trabajar en la práctica con esquemas completamente homomórficos, tanto el que se ha presentado en este capítulo como los mencionados en 2.3.3, existen una series de librerías para diferentes lenguajes de programación que los implementan con mayor eficiencia, y considerando estándares de seguridad informática que sobrepasan el alcance de la implementación didáctica presentada.

Se ha utilizado como referencia el repositorio [23] donde se mantiene una lista actualizada de librerías y herramientas relacionadas con los esquemas homomórficos.

Entre las librerías disponibles caben destacar `OpenFHE`, de código abierto, la cual imprenta en el lenguaje `C++` los principales esquemas completamente homomórficos tales como: BGV, BFV, CKKS y otros que no se han mencionado anteriormente tales como DM (Ducas-Micciancio) y CGGI (Chillotti-Gama-Georgieva-Izabachene).

Otra librería, parcialmente incluida es la anterior, es `HEAAN`, de código abierto y desarrollada en `C++`, que implementa la aritmética de números aproximada empleada en el esquema CKKS y permite además emplear el recifrado en este.

Como se ve, muchas de las implementaciones disponibles se encuentran desarrolladas en `C++` debido a la mejora de rendimiento que aporta, no obstante también hay disponibles librerías en otros lenguajes como `C, Python, Go` y `JavaScript`, entre otros.

En la práctica, el uso de esquemas de encriptación homomórficos en aplicaciones informáticas lo podemos encontrar a través de *frameworks* que, haciendo uso de estas librerías, facilitan tareas como la minería de datos sobre conjuntos de datos cifrados; o aportando confidencialidad a los datos utilizados en modelos de aprendizaje automático.

Para esta última tarea podemos mencionar `TF Encrypted`, un *framework* que permite trabajar con la popular librería de aprendizaje automático `TensorFlow` con datos previamente cifrados. También cabe mencionar el *framework* `Rosetta` que, al igual que el mencionado anteriormente, permite trabajar con `TensorFlow` añadiendo además la posibilidad de usar protocolos de computación segura multipartita, con el fin de descentralizar los cálculos realizados de forma segura.

Conclusiones

Tras haber explorado los principales conceptos dentro de la Criptografía Homomórfica, podemos ver las aportaciones que puede llegar a ofrecer. Entre ellas, facilitar la descentralización del procesado de información confidencial, delegando este en terceras partes que no pondrían acceder a la información propiamente, pero serán capaces de realizar cálculos con esta.

Otro uso práctico que este tipo de esquemas puede llegar a ofrecer se encuentra en el caso en el que se desee mantener la privacidad tanto de los datos, como de los algoritmos que describan los cálculos realizados sobre estos por una tercera parte.

Algunas aplicaciones de este tipo de esquemas se encuentran en áreas donde la privacidad es de mayor importancia, tales como la medicina, el minado de datos, o el aprendizaje automático. Por ejemplo, mediante el cálculo de indicadores comunes para enfermedades a partir de resultados de pruebas médicas cifrados.

Además, tras haber estudiado los principales problemas con retículos y como se establecen algunos esquemas de encriptación a partir de ellos, podemos tener una pequeña visión sobre el tipo de esquemas que se plantean seguros en la actualidad para la Computación Cuántica.

Por último, gracias a la implementación realizada, se ha visto de forma práctica el elevado número de operaciones que conlleva evaluar circuitos homomórficamente, siendo el consumo de recursos una de las principales desventajas de estos esquemas.

Para concluir esta breve introducción al campo de la Criptografía Homomórfica, se debe resaltar la gran utilidad que presenta el esquema CKKS, el cual no se ha llegado a describir propiamente, pero es capaz de agilizar la evaluación de circuitos a costa de la precisión en el cálculo, presentando un interesante camino a estudiar.

Bibliografía

[1] Buchmann, Johannes A.: *Introduction to Cryptography*. Undergraduate Texts in Mathematics. Springer, 2001, ISBN 978-1-4684-0498-2.

[2] Information Technology Laboratory (NIST): *Announcing the Advanced Encryption Standard (AES)*. Information Technology Laboratory, NIST, 2001. https://www.nist.gov/publications/advanced-encryption-standard-aes, visitado el 2022-02-10.

[3] Rivest, R. L., A. Shamir y L. Adleman: *A method for obtaining digital signatures and public-key cryptosystems*. Communications of the ACM, 21(2):120–126, 1978, ISSN 0001-0782, 1557-7317.

[4] Goldwasser, Shafi y Silvio Micali: *Probabilistic encryption*. Journal of Computer and System Sciences, 28(2):270–299, 1984, ISSN 00220000.

[5] Elgamal, T.: *A public key cryptosystem and a signature scheme based on discrete logarithms*. IEEE Transactions on Information Theory, 31(4):469–472, 1985, ISSN 0018-9448.

[6] Diffie, W. y M. Hellman: *New directions in cryptography*. IEEE Transactions on Information Theory, 22(6):644–654, 1976, ISSN 0018-9448.

[7] Lenstra, A. K., H. W. Lenstra y L. Lovász: *Factoring polynomials with rational coefficients*. Mathematische Annalen, 261(4):515–534, 1982.

[8] Regev, Oded: *On lattices, learning with errors, random linear codes, and cryptography*. Journal of the ACM, 56(6):1–40, 2009, ISSN 0004-5411, 1557-735X.

[9] Lyubashevsky, Vadim, Chris Peikert y Oded Regev: *On Ideal Lattices and Learning with Errors over Rings*. Journal of the ACM, 60(6):1–35, 2013.

[10] Adeline Langlois, Damien Stehlé: *Hardness of decision (R)LWE for any modulus*. Cryptology ePrint Archive, Paper 2012/091, 2012. https://ia.cr/2012/091, visitado el 2022-03-02.

[11] Goldreich, Oded, Shafi Goldwasser y Shai Halevi: *Public-key cryptosystems from lattice reduction problems*. En *Advances in Cryptology — CRYPTO '97*, volumen 1294, páginas 112–131. Springer Berlin Heidelberg, 1997.

[12] Micciancio, Daniele: *Improving Lattice Based Cryptosystems Using the Hermite Normal Form*. En *Cryptography and Lattices*, volumen 2146 de *Lecture Notes in Computer Science*, páginas 126–145. Springer Berlin Heidelberg, 2001, ISBN 978-3-540-42488-8.

[13] Nguyen, Phong: *Cryptanalysis of the Goldreich-Goldwasser-Halevi Cryptosystem from Crypto '97*. En *Advances in Cryptology — CRYPTO' 99*, volumen 1666 de *Lecture Notes in Computer Science*, páginas 288–304. Springer Berlin Heidelberg, 1999.

[14] Gentry, Craig: *Fully homomorphic encryption using ideal lattices*. En *Proceedings of the 41st annual ACM symposium on Symposium on theory of computing - STOC '09*, página 169. ACM Press, 2009.

[15] Gentry, Craig y Shai Halevi: *Implementing Gentry's Fully-Homomorphic Encryption Scheme*. En *Advances in Cryptology – EUROCRYPT 2011*, volumen 6632 de *Lecture Notes in Computer Science*, páginas 129–148. Springer Berlin Heidelberg, 2011, ISBN 978-3-642-20464-7.

[16] Smart, N. P. y F. Vercauteren: *Fully Homomorphic Encryption with Relatively Small Key and Ciphertext Sizes*. En *Public Key Cryptography – PKC 2010*, volumen 6056, páginas 420–443. Springer Berlin Heidelberg, 2010.

[17] Armknecht, F., C. Boyd, C Carr y K Gjøsteen et al: *A Guide to Fully Homomorphic Encryption*. Cryptology ePrint Archive, Paper 2015/1192, 2015. https://ia.cr/2015/1192, visitado el 2022-04-01.

[18] Kim, Andrey, Yuriy Polyakov y Vincent Zucca: *Revisiting Homomorphic Encryption Schemes for Finite Fields*. Cryptology ePrint Archive, Paper 2021/204, 2021. https://ia.cr/2021/204, visitado el 2022-07-25.

[19] Cheon, J. H., A. Kim, M. Kim y Y. Song: *Homomorphic Encryption for Arithmetic of Approximate Numbers*. En *Advances in Cryptology – ASIACRYPT 2017*, volumen 10624 de *Lecture Notes in Computer Science*, páginas 409–437. Springer International Publishing, 2017.

[20] Fan, Junfeng y Frederik Vercauteren: *Somewhat Practical Fully Homomorphic Encryption*. Cryptology ePrint Archive, Paper 2012/144, 2012. https://ia.cr/2012/144, visitado el 2022-04-20.

[21] Python Core Team: *Python: A dynamic, open source programming language*. Python Software Foundation, 2021. https://www.python.org/, Python version 3.10.

[22] Harris, Charles R, K Jarrod Millman y Stéfan J van der Walt et al.: *Array programming with NumPy*. Nature, 7825:357–362, 2020, ISSN 1476-4687.

[23] Jonathan Schneider et al.: *Awesome Homomorphic Encryption: A curated list of amazing Homomorphic Encryption libraries, software, and resources*. Repositorio en GitHub, 2017. https://github.com/jonaschn/awesome-he, visitado el 2022-05-10.

Introduction to Homomorphic Cryptography

Pedro Gonzalo Méndez Hernández
Facultad de Ciencias • Sección de Matemáticas
Universidad de La Laguna
alu0100883565@ull.edu.es

Abstract

*This work will give a brief introduction to Cryptography and the main concepts and mathematical tools used for the description of encryption schemes. Most significantly, the concept of **homomorphic cryptography** will be presented and a brief review of its evolution will be made. Moreover, the main cryptographic elements on which the security of homomorphic schemes are based, such as the problems in **ideal lattices**, will be described.*
*Lastly, a Python implementation of the **BFV** homomorphic encryption scheme will be provided.*

1. Introduction

Cryptography is the study of mathematical techniques related to aspects of information security, such as confidentiality, data integrity, authentication and non-repudiation.

Encryption schemes are the cryptographic tools used to provide information confidentiality, preventing third parties from obtaining access to encrypted information. These schemes are designed based on **cryptographic primitives** whose properties and security are proven, and algorithms are developed around them to provide information confidentiality. The resistance of these schemes to attacks by third parties relies on the computational complexity assumptions derived from the chosen primitives.

An encryption scheme $\mathcal{E}$ consist of the tuple $(\mathcal{P}, \mathcal{C}, \mathcal{K}, \mathcal{ENC}, \mathcal{DEC})$ where:

- $\mathcal{P}$, is the set of every m plaintext, that can be encrypted with the scheme
- $\mathcal{C}$, is the set of all possible c ciphertexts
- $\mathcal{K}$, the set of all k_i necessaries for encryption K_{Enc} and decryption K_{Dec}.
- $\mathcal{ENC} = \{Enc_k : k \in \mathcal{K}_{Enc}\}$, where $Enc_k : \mathcal{P} \longrightarrow \mathcal{C}$ are the encryption functions.
- $\mathcal{DEC} = \{Dec_k : k \in \mathcal{K}_{Dec}\}$, where $Dec_k : \mathcal{C} \longrightarrow \mathcal{P}$ are the decryption functions.

Furthermore, $\forall e \in K_{Enc}\ \exists d \in K_{Dec} : Dec_d(Emc_e(m)) = m\ \forall m \in \mathcal{P}$.

Homomorphic Cryptography is the field of Cryptography that covers homomorphic encryption schemes. These schemes are those that allow a set of operations to be performed on the ciphertext so that the result is still encrypted and, when decrypted, we obtain the result of this operation in plain text. Formally, we have for an internal binary operation $F_{\mathcal{P}}$ on the plaintext domain, and an associated operation $F_{\mathcal{C}}$ on the ciphertext domain such that:

$$F_{\mathcal{P}}(m_1, m_2) = Dec_d(F_{\mathcal{C}}(Enc_e(m_1), Enc_e(m_2))) : \\ m_1, m_2 \in \mathcal{P}, d \in K_{Dec}, e \in K_{Enc}.$$

Schemes that satisfies this for only one F are called **partially homomorphic** schemes, and those that have two operations satisfying this for a limited number of operation are called **somewhat homomorphic**. Meanwhile, schemes that have two operations verifying it for any number of consecutive operations are **fully homomorphic** schemes.

2. Outline

In this work, we will revisit some of the most used cryptographic elements on homomorphic schemes. Particularly, the computational hardness assumptions of the ideal lattices problems such as the **Shortest Vector Problem** (SVP) or the Closest Vector Problem (CVP). Moreover, we will discuss the Learning with Errors (LWE) and Learning with Errors over Rings (RLWE) problems. The BFV scheme, which we will describe and implement, uses the RLWE as a cryptographic primitive to ensure the security of the scheme. The **search RLWE** problem can be summarized as:

> Let $n = 2^k$ for a $k \in \mathbb{N}$ and $q \in \mathbb{N}$. Consider $s(x) \in \mathcal{A}_q = \mathbb{Z}_q[x]/(x^n + 1)$ and m independently sampled $e_i(x)$ from a distribution χ on $\mathcal{A} = \mathbb{Z}[x]/(x^n + 1)$ bounded and m samples taken independently from $a_i(x) \longleftarrow U(\mathcal{A})$

The problem goal is obtaining $s(x)$ from the m tuples $(a_i(x), b_i(x) = [a(x) \cdot s(x) + e_i(x))]_q$.
Some instances of this problem can be reduced to the SVP, a problem for which there is no algorithm solving it in polynomial time.

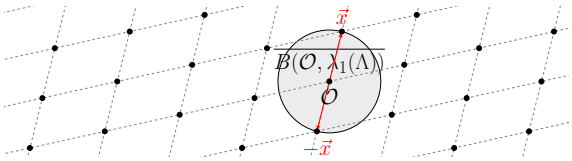


Figure 1: *In this example, both $\vec{x}$ and $-\vec{x}$ solves the SVP for this lattice Λ. Where $\lambda(\Lambda)$ is its length.*

3. BFV scheme

Designed by J. Fan and F. Vercauterense, and based on a similar cryptosystem provided by Bakerski. This scheme relies on the RLWE problem previously mentioned and the **bootstrapping method** for somewhat homomorphic schemes, presented by Gentry, to stablish a scheme capable of homomorphically evaluate circuits of the desired depth securely.

4. Other homomorphic schemes and its applications

Besides the BFV scheme, there are other fully homomorphic schemes mostly based on the ideal lattices problems such as the **BGV** (Brakerski, Gentry and Vaikuntanatha) scheme or other based on the **NTRU** cryptosystem.
Most significantly, the **CKKS** (Cheon, Kim, Kim, and Song) scheme which relies on arithmetic of approximate numbers to provide a fully homomorphic scheme, trading computation precision for performance. has proven to be a promising tool in the fields of Data Mining and Machine Learning. This scheme provides confidentiality for the massive amounts of valuable information usually used to produce analysis and develop models in these fields.

References

[1] BUCHMANN, Johannes A.: *Introduction to Cryptography* Undergraduate Texts in Mathematics. Springer, 2001.

[2] ARMKNECHT, F. et al: *A Guide to Fully Homomorphic Encryption* Cryptology ePrint Archive, Paper 2015/1192, 2015. https://ia.cr/2015/1192 visited on 2022-04-01.

[3] FAN, Junfeng and VERCAUTEREN, Frederik: *Somewhat Practical Fully Homomorphic Encryption* Cryptology ePrint Archive, Paper 2012/144, 2012. https://ia.cr/2012/144 visited on 2022-04-20.