Escuela Técnica Superior de Ingeniería Informática

Grado en Matemáticas

Curso Académico 2019/2020

Trabajo Fin de Grado

Análisis de un esquema de cifrado construido sobre Cuasigrupos

Alejandro García Carretero

Directora:

María Isabel González Vasco

Índice general

Capítulo 1

Objetivos.

Los objetivos principales que se buscan alcanzar en este trabajo de investigación son los siguientes:

1. Abordar un tema que no ha sido tratado en las asignaturas cursadas durante el grado, o que no se ha estudiado con suficiente profundidad, como es la criptografía.
2. Completar y afianzar contenidos vistos a lo largo de la carrera, en este caso asociados a las estructuras algebraicas; así como estudiar su papel y aplicación en el ámbito de la criptografía.
3. Realizar un estudio riguroso acerca de un esquema criptográfico concreto, aportando una exposición clara y descriptiva del mismo, así como explicando sus fundamentos y observando cómo se comporta con diferentes configuraciones.
4. Aportar un enfoque crítico y creativo del esquema estudiado, buscando posibles fallos o errores y ofreciendo propuestas para solucionar los mismos, sugiriendo distintas variantes y alternativas al sistema original.

Capítulo 2

Introducción.

La criptografía es, según la RAE, el "arte de escribir con clave secreta o de un modo enigmático", aunque de modo más formal puede definirse como la ciencia o técnica de la protección de información. Lo que está claro es que la criptografía lleva con nosotros mucho tiempo, desde el primer cifrado conocido en el siglo V a.C, pasando por los cifrados clásicos como el cifrado César, hasta las máquinas Enigma alemanas de la II Guerra Mundial, que fueron criptanalizadas por grandes matemáticos como Marian Rejewski o Alan Turing, considerado el padre de la computación. No fue precisamente hasta el siglo XX que se empezaron a hacer grandes avances en este campo gracias a la llegada de los ordenadores y, con ellos, el abordaje de tareas computacionales de elevada complejidad.

Hoy en día, desde la llegada de Internet, la criptografía es algo que nos acompaña a todos a diario, y es un elemento crucial en la ciberseguridad de los sistemas de todo el mundo. Proporciona medios para proteger el almacenamiento de datos y la transmisión de información en infinidad de escenarios, desde los datos contenidos en un teléfono móvil, hasta contraseñas de servicios almacenadas en servidores, incluyendo transacciones bancarias que transitan por la red. Es una necesidad, puesto que nuestros datos nos definen e identifican, que esta información no sea accesible a terceras personas no autorizadas que puedan querer hacer un mal uso de ella.

En el ámbito científico, la criptografía está también en auge. Cada vez son más las empresas y estados que ponen más énfasis y presupuesto en la investigación y desarrollo de esta rama de la matemática aplicada. Esto se debe a que el progreso tecnológico es muy rápido, y los esquemas criptográficos existentes se van quedando obsoletos, y son expuestos o comprometidos gracias a los avances regulares en computación.

Personalmente, la ciberseguridad y criptografía siempre me han parecido de gran interés. Creo que en la sociedad de hoy en día, los datos que generamos aportan toda la información necesaria para saber quiénes somos, con quién nos relacionamos, nuestros gustos, nuestros intereses, ... incluso información que nos caracteriza y de cuya existencia explícita ni siquiera

somos conscientes. Debido a esto, ahora son más importantes que nunca la protección y el control de dicha información, así como su confidencialidad, integridad y disponibilidad. A medida que iba profundizando en conocimientos sobre informática y matemáticas, me iba dando cuenta de que es un área que tiene múltiples aplicaciones, y en el que hay mucho que investigar e implementar. Dado que a lo largo de la carrera no tenemos ninguna asignatura sobre criptografía, y siendo ésta un área que me llama mucho la atención por conjugar conceptos tanto matemáticos cómo informáticos, he decidido introducirme en la materia realizando una investigación sobre aspectos criptográficos, aplicando conocimientos teóricos del ámbito del álgebra y de la estadística.

En este trabajo he estudiado y analizado un esquema de cifrado basado en cuasigrupos presentado por unos investigadores de las universidades de Zielona Góra, Polonia y de Pensilvania, Estados Unidos. Este análisis se ha realizado ejemplificando y explicando su comportamiento en distintas situaciones y sobre diferentes implementaciones en distintas estructuras algebraicas. Además, este estudio crítico se acompaña de propuestas de solución a los fallos identificados, así como introducciones teóricas, tanto en el ámbito más puramente matemático, como más concretamente criptográfico.

Durante la elaboración del proyecto he desarrollado unas herramientas de apoyo para facilitar los cálculos sobre cuasigrupos y cuerpos finitos con distintas utilidades sobre los cifrados que se estudian a lo largo del mismo. El código que implementa de dichas herramientas ha sido escrito en lenguaje Python y se encuentra como anexo al trabajo, documentado con una breve explicación acerca de su uso y funcionamiento.

Capítulo 3

Fundamentos Matemáticos.

En este capítulo se hará una breve introducción de las principales definiciones y propiedades de cuasigrupos, grupos, anillos y cuerpos. Así, se resumen los fundamentos básicos para poder entender los resultados y esquemas criptográficos que se presentan más adelante. Esta sección sigue esencialmente lo presentado en [9] y [10], con ampliaciones de [1] y [3].

3.1. Estructuras algebraicas básicas.

Definición 3.1.1 Cuasigrupo. Un *cuasigrupo* $(Q, \oplus)$ es una estructura algebraica que consta de un conjunto Q, dotado de una ley de composición interna

$$\begin{array}{rccc} \oplus: & Q \times Q & \longrightarrow & Q \\ & (x, y) & \mapsto & x \oplus y. \end{array}$$

que cumple, $\forall a, b \in Q$:

$$1. \quad \exists! \quad c \in Q \quad | \quad a \oplus c = b$$

$$2. \quad \exists! \quad d \in Q \quad | \quad d \oplus a = b.$$

Si Q tiene pocos elementos, podemos representar $\oplus$ en forma de tabla, de forma que $x \oplus y$ es el cruce de la fila de x con la columna de y. Debido a estas condiciones, en las tablas resultantes no se repite ningún elemento de Q ni en las filas ni en las columnas.

A todos los cuasigrupos se les asocia, adicionalmente, dos operaciones más que son derivadas de forma natural de $\oplus$. Son las operaciones binarias "división por la derecha" (/) y "división por la izquierda" (\ ), que cumplen:

1. $y\backslash(y \oplus x) = x$
2. $y \oplus (y\backslash x) = x$
3. $(x \oplus y)/y = x$
4. $(x/y) \oplus y = x$.

Definición 3.1.2 Anillo. Un *anillo* $(A, +, *)$ es una estructura algebraica formada por un conjunto A y dos operaciones binarias internas que cumplen:

1. $\forall a, b, c \in A, \quad a + (b + c) = (a + b) + c$ [+ Asociativa]
2. $\exists e \in A \mid \forall a \in A \quad a + e = e + a = a$ [+ Neutro]
3. $\forall a \in A \quad \exists b \in G \mid a + b = b + a = e$ [+ Opuestos]
4. $\forall a, b \in A \quad a + b = b + a$ [+ Conmutativa]
5. $\forall a, b, c \in A \quad (a * b) * c = a * (b * c)$ [* Asociativo]
6. $\begin{cases} \forall a, b, c \in A \quad a * (b + sc) = a * b + a * c \\ \forall a, b, c \in A \quad (b + c) * a = b * a + c * a \end{cases}$ [* Distributivo respecto de +]

Definición 3.1.3 Anillo conmutativo. Un *anillo* se dice *conmutativo* si $*$ es conmutativa:

$$\forall a, b \in A \quad a * b = b * a.$$

Definición 3.1.4 Anillo unitario. Un *anillo* se dice que es *unitario* o *con unidad*, si para la operación $*$ tiene elemento neutro:

$$\exists u \in A, u \neq e \mid \forall a \in A \quad a * u = u * a = a.$$

En un anillo unitario, dado un $a \in A$, decimos que a tiene inverso, o que a es invertible, si

$$\exists b \in A \mid a * b = b * a = u.$$

Definición 3.1.5 Característica de un anillo. Llamamos *característica de un anillo* A al número $n \in \mathbb{N}$ más pequeño que cumple $n * x = e \ \forall x \in A$. Si no existe dicho n, entonces se dice que la característica del anillo A es 0.

Definición 3.1.6 Cuerpo. Un *cuerpo* es un anillo conmutativo con unidad, tal que todo elemento distinto de e tiene inverso.

Definición 3.1.7 Espacio Vectorial. Sean V un conjunto arbitrario y K un cuerpo. Llamamos *espacio vectorial sobre* K a la estructura formada por el conjunto V y dotada de dos

operaciones [7]:

$$+: \begin{array}{ccc} V \times V & \longrightarrow & V \\ (u,v) & \mapsto & u+v=w \end{array} \qquad \cdot: \begin{array}{ccc} K \times V & \longrightarrow & V \\ (\lambda,v) & \mapsto & \lambda \cdot v = w. \end{array}$$

que cumplen las siguientes propiedades:

1. $u+v=v+u \quad \forall u,v \in V$
2. $u+(v+w)=(u+v)+w \quad \forall u,v,w \in V$
3. $\exists e \in V$ tal que $v+e=e \quad \forall v \in V$
4. $\forall v \in V \quad \exists -v \in V$ tal que $v+(-v)=e$
5. $a \cdot (b \cdot v) = (a \cdot b) \cdot v \quad \forall a,b \in K, \quad v \in V$
6. $\exists e \in K$ tal que $v \cdot e = v \quad \forall v \in V$
7. $a \cdot (u+v) = a \cdot u + b \cdot v \quad \forall a,b \in K \quad u,v \in V$
8. $(a+b) \cdot v = a \cdot v + b \cdot v \quad \forall a,b \in K \quad v \in V$.

Definición 3.1.8 Grupo. Un *grupo* $(G, \oplus)$ es una estructura algebraica que también está formada por un conjunto y una operación binaria interna, que cumple que:

1. $$\oplus: \begin{array}{ccc} G \times G & \longrightarrow & G \\ (a,b) & \mapsto & a \oplus b \end{array}$$
2. $\forall a,b,c \in G, \quad a \oplus (b \oplus c) = (a \oplus b) \oplus c$ [Asociativa]
3. $\exists e \in G \mid \forall a \in G \quad a \oplus e = e \oplus a = a$ [Neutro]
4. $\forall a \in G \quad \exists b \in G \mid a \oplus b = b \oplus a = e$ [Inversos]

Definición 3.1.9 Grupo abeliano. Si además se cumple $\forall a,b \in G, \quad a \oplus b = b \oplus a$ [Conmutativa], entonces se dice que G es *conmutativo* o *abeliano*.

Propiedades 3.1.10 Se cumplen, además, una serie de propiedades:

1. El elemento neutro es único
2. Los inversos son únicos
3. Propiedad de cancelación: $\begin{cases} a \oplus b = a \oplus c \Rightarrow b = c \\ b \oplus a = c \oplus a \Rightarrow b = c \end{cases}$.

Observación 3.1.11 Todo grupo es un cuasigrupo. Se puede demostrar fácilmente que cualquier grupo cumple la definición 3.1.1:

1. Suponiendo que existen c_1 y c_2 tales que $a \oplus b = c_1$ y que $a \oplus b = c_2$, y siendo a' tal que $a \oplus a' = e$, tenemos que usando la definición y la propiedad de cancelación:

$$\begin{cases} a' \oplus c_1 = a' \oplus (a \oplus b) = (a' \oplus a) \oplus b = e \oplus b = b \\ a' \oplus c_2 = a' \oplus (a \oplus b) = (a' \oplus a) \oplus b = e \oplus b = b \end{cases} \Rightarrow a' \oplus c_1 = a' \oplus c_2 \Rightarrow c_1 = c_2.$$

2. Suponiendo que existen d_1 y d_2 tales que $d_1 \oplus a = b$ y que $d_1 \oplus a = b$, tenemos que usando la propiedad de cancelación:

$$\begin{cases} d_1 \oplus a = b \\ d_2 \oplus a = b \end{cases} \Rightarrow d_1 \oplus a = d_2 \oplus a \Rightarrow d_1 = d_2.$$

Definición 3.1.12 Grupos de permutaciones. Dado un conjunto finito T llamamos *grupo simétrico asociado a* T (denotado por S_T) al grupo formado por las aplicaciones biyectivas de T en sí mismo.

Es fácil ver que los grupos de permutaciones se caracterizan por el cardinal del conjunto sobre el que actúan, así, si $\#T = n$, en lugar de escribir S_T denotamos por S_n al grupo simétrico asociado, pues todos los grupos de permutaciones actuando sobre conjuntos de n elementos son isomorfos.

3.2. Ejemplos de cuasigrupos.

Ejemplo 3.2.1 Cuadrado latino. Dados n elementos, un *cuadrado latino* es una matriz cuadrada de dimensiones $n \times n$ compuesta por dichos elementos, distribuidos de forma que ninguno se repite ni en la misma fila, ni en la misma columna [9]. Por ejemplo, sea el siguiente cuadrado latino:

$$\begin{matrix} 0 & 2 & 1 \\ 1 & 0 & 2 \\ 2 & 1 & 0 \end{matrix}$$

Dada la operación $\oplus$, y el cuadrado latino anterior, podemos montar la tabla de la operación y observar cómo se cumple la definición 3.3.1. Esto es consecuencia de no repetir elementos en filas ni columnas.

$\oplus$	0	1	2
0	0	2	1
1	1	0	2
2	2	1	0

Ejemplo 3.2.2 Sistema de Steiner. Un ejemplo de *sistema triple de Steiner* es el espacio

proyectivo sobre $\mathbb{Z}_2$ que da lugar al Plano de Fano. Lo que nos deja los conjuntos:

$$S = \{1, 2, 3, 4, 5, 6, 7\}.$$

$$\mathcal{B} = \{\{2, 4, 6\}, \{1, 4, 5\}, \{3, 4, 7\}, \{1, 2, 3\}, \{2, 5, 7\}, \{1, 6, 7\}, \{3, 5, 6\}\}.$$

Donde cada elemento de $\mathcal{B}$ es una linea que pasa por los 3 puntos que forman el bloque.

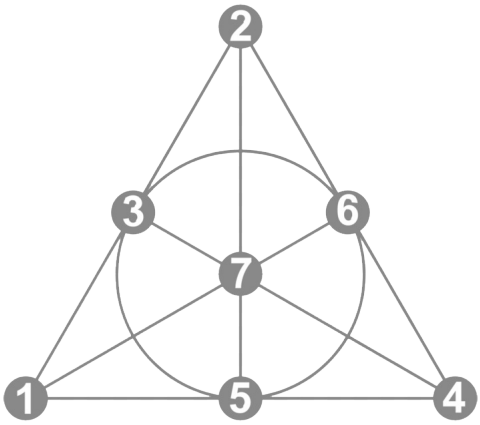

Figura 3.1. Plano de Fano

Un sistema triple de Steiner forma un cuasigrupo con la operación $\oplus$ si cumple la propiedad:

$$x \oplus y = z \iff \{x, y, z\} \in \mathcal{B} \vee x = y = z$$

.

Como consecuencia, se cumplen 2 propiedades adicionales:

1. Es un cuasigrupo idempotente: $x \oplus x = x \quad \forall x \in S$
2. Tiene simetría total: $x \oplus y = x/y = x \backslash y$.

Ejemplo 3.2.3 Espacios vectoriales sobre cuerpos finitos. Un *hespacio vectorial V construido sobre un cuerpo finito* con característica distinta de 2, forma un cuasigrupo con la operación:

$$\begin{array}{rccc} \oplus: & V \times V & \longrightarrow & V \\ & (u, v) & \mapsto & \frac{u+v}{2}. \end{array}$$

Por ejemplo, tomando $\mathbb{Z}_2$ como cuerpo, podemos formar un espacio vectorial V de dimensión n. Entonces sus elementos serían $v \in V \mid v = (v_1 \dots v_n)$ donde $v_i \in \mathbb{Z}_2 \quad \forall i \in \{1 \dots n\}$.

Ejemplo 3.2.4 Grupos de enteros modulo n. El conjunto de los *enteros modulo n* $(\mathbb{Z}_n, \oplus)$ con la suma habitual modulo n forma un grupo. Por ejemplo $(\mathbb{Z}_5, \oplus)$, donde podemos ver en la tabla de $\oplus$ que se cumplen todas las propiedades, y que además es abeliano.

$\oplus$	0	1	2	3	4
0	0	1	2	3	4
1	1	2	3	4	0
2	2	3	4	0	1
3	3	4	0	1	2
4	4	0	1	2	3

Ejemplo 3.2.5 Grupos de permutaciones. Por ejemplo, S_3 es el *grupo de todas las permutaciones sobre* $\mathbb{Z}_3$, y sus elementos son:

$$S_3 = \{id, f, g, h, i, j\}.$$

$$\begin{array}{rccc} id: & \mathbb{Z}_3 & \longrightarrow & \mathbb{Z}_3 \\ & 0 & \mapsto & 0 \\ & 1 & \mapsto & 1 \\ & 2 & \mapsto & 2 \end{array} \qquad \begin{array}{rccc} f: & \mathbb{Z}_3 & \longrightarrow & \mathbb{Z}_3 \\ & 0 & \mapsto & 1 \\ & 1 & \mapsto & 2 \\ & 2 & \mapsto & 0 \end{array}$$

$$\begin{array}{rccc} g: & \mathbb{Z}_3 & \longrightarrow & \mathbb{Z}_3 \\ & 0 & \mapsto & 2 \\ & 1 & \mapsto & 0 \\ & 2 & \mapsto & 1 \end{array} \qquad \begin{array}{rccc} h: & \mathbb{Z}_3 & \longrightarrow & \mathbb{Z}_3 \\ & 0 & \mapsto & 0 \\ & 1 & \mapsto & 2 \\ & 2 & \mapsto & 1 \end{array}$$

$$\begin{array}{rccc} i: & \mathbb{Z}_3 & \longrightarrow & \mathbb{Z}_3 \\ & 0 & \mapsto & 2 \\ & 1 & \mapsto & 1 \\ & 2 & \mapsto & 0 \end{array} \qquad \begin{array}{rccc} j: & \mathbb{Z}_3 & \longrightarrow & \mathbb{Z}_3 \\ & 0 & \mapsto & 1 \\ & 1 & \mapsto & 0 \\ & 2 & \mapsto & 2. \end{array}$$

Observación 3.2.1 Notación. Se puede emplear la notación de ciclos, donde por ejemplo $g = (0 \quad 2 \quad 1)$. Esto quiere decir que a través de g el 0 se manda al 2, el 2 se manda al 1 y el 1 se manda al 0. Si la permutación deja quieto algún elemento del conjunto original, este no aparece en la notación del ciclo, como por ejemplo: $j = (0 \quad 1)$.

Los conjuntos de las permutaciones forman grupo con la operación 'composición' $(S_n, \circ)$

La forma de componer las permutaciones es la composición de funciones correspondiente, así:

$$g \circ j = (0 \quad 2 \quad 1) \circ (0 \quad 1) = (1 \quad 2) = h.$$

Se puede ver como composición de funciones de forma sencilla:

$$\begin{array}{rccccc} h = g \circ j: & \mathbb{Z}_3 & \overset{\text{j}}{\to} & \mathbb{Z}_3 & \overset{\text{g}}{\to} & \mathbb{Z}_3 \\ & 0 & \mapsto & 1 & \mapsto & 0 \\ & 1 & \mapsto & 0 & \mapsto & 2 \\ & 2 & \mapsto & 2 & \mapsto & 1. \end{array}$$

Capítulo 4

Criptografía: Breve introducción.

Popularmente se denomina criptografía a todo el estudio relacionado con el intercambio de mensajes, claves secretas, etc. , pero lo correcto es denominarlo criptología. La criptología tiene distintas patas: entre ellas la criptografía, que se encarga del estudio "constructivo" o diseño de los algoritmos y cifrados; y el criptoanálisis, cuyo objetivo es estudiar las vulnerabilidades y ataques a los esquemas y protocolos criptográficos.

En esta sección vamos a hacer una breve introducción a la criptografía y el criptoanálisis con el fin de facilitar y resumir los conceptos que se usarán más tarde en la explicación del esquema y su posterior análisis. Estudiaremos la diferencia entre los distintos tipos de sistemas de cifrado y algunos de los posibles ataques que pueden llevarse a cabo para probar la seguridad o fiabilidad de los mismos.

El objetivo de los esquemas de cifrado es que dos o más partes puedan realizar un intercambio seguro de mensajes o textos planos, evitando que terceros puedan recuperar total o parcialmente estos mensajes. De esta forma, para enviar un mensaje lo ciframos y para poder leerlo, lo desciframos.

Informalmente, podemos dar una definición como sigue: un algoritmo de cifrado transforma, bajo una clave determinada, un texto claro en un texto cifrado. Lo denotamos con $c = \mathcal{E}(k, m)$. Donde c es el texto cifrado, m el texto claro, y $\mathcal{E}$ el algoritmo de cifrado que usa la clave k.

Al proceso inverso, mediante el cual recuperamos el texto claro a partir del texto cifrado utilizando una clave k', lo denominamos algoritmo de descifrado, y se denota $m = \mathcal{D}(k', c)$.

Observación. $\mathcal{E}(k, m)$ y $\mathcal{D}(k', c)$ no son funciones, sino algoritmos, es decir, procesos.

Principio de Kerckhoffs. Cabe destacar que a la hora de hablar de sistemas de cifrado, la descripción de los algoritmos concretos que se emplean han de considerarse públicos,

siguiendo el *Principio de Kerckhoffs*. El Principio de Kerckhoffs dice que

"El método de cifrado no deber ser secreto, y que caiga en manos de un enemigo no debe causar ningún problema".

Dicho de otra manera, la seguridad de un esquema de cifrado no debe basarse en ocultar sus algoritmos. Es decir, si los algoritmos de cifrado y descifrado son conocidos por un atacante, el esquema no debe quedar comprometido. Publicar los algoritmos tiene además una serie de ventajas:

1. Es más fácil mantener en secreto únicamente una clave pequeña, que un algoritmo entero.

2. En caso de exposición del secreto, es más fácil cambiar una clave que todo un esquema de cifrado, ya que diseñar un nuevo esquema es mucho más costoso que volver a generar claves aleatorias.

3. En despliegues a gran escala, es más sencillo y eficaz, tanto a nivel algorítmico como de software, reutilizar los esquemas para las distintas comunicaciones, cambiando únicamente las claves. Esto permite la estandarización de los esquemas, asegurando así grandes compatibilidades y facilidad de comunicación.

4. Publicar los algoritmos permite detectar vulnerabilidades más rápido. Esto se traduce en mejoras de los esquemas y en la utilización de esquemas de cifrado a los que no se les haya encontrado fallos de seguridad.

4.1. Esquemas de cifrado

4.1.1. Cifrado de clave privada

Los sistemas simétricos o de clave privada son aquellos en los que se usa una misma clave, tanto para cifrar como para descifrar los mensajes, es decir, $k = k'$. Por tanto, ésta debe ser conocida por ambas partes de la comunicación. A la vez, esto puede ser un problema a la hora de la distribución de la clave, ya que si se intercepta durante ese proceso, la comunicación queda comprometida. Por ello, se supondrá en los sistemas de clave privada que tratemos que la generación y distribución de la clave se realiza por un canal seguro y correcto.

La información pública en esos sistemas es únicamente los algoritmos de cifrado $\mathcal{E}(k, m)$ y descifrado $\mathcal{D}(k, c)$, mientras que la clave k permanece en secreto. De esta forma queda un esquema de la comunicación de este modo:

$$\begin{array}{ccc} \mathcal{A} & \longrightarrow & \mathcal{B} \\ m & & \\ c = \mathcal{E}(k, m) & \longrightarrow & c \\ & & m = \mathcal{D}(k, c) \end{array}$$

Formalmente, se definen los esquemas de clave privada como sigue.

Definición 4.1.1 Sistema de clave privada. Un *sistema de clave privada* $\mathcal{SE} = (\mathcal{G}, \mathcal{E}, \mathcal{D})$ se define como el conjunto de 3 algoritmos: [2]

1. El algoritmo de generación de claves $\mathcal{G}$, que devuelve una clave k. Denotamos $\mathcal{K}$ al conjunto de todas las cadenas que tienen probabilidad de ser la salida de $\mathcal{G}$, y lo denominamos espacio de claves. Lo denotamos por $k \leftarrow \mathcal{G}(1^n)$, $k \in \mathcal{G}(1^n)$ o $k \in \mathcal{K}$.
2. El algoritmo de cifrado $\mathcal{E}$, que puede ser aleatorizado o determinista, actúa sobre el espacio de mensajes $\mathcal{M}$. Toma una clave $k \in \mathcal{K}$ y un texto claro $m \in \mathcal{M}$ para devolver un texto cifrado $c \in \mathcal{C} \cup \{\perp\}$, donde $\perp$ representa un mensaje de error como se detalla en la siguiente observación. Lo denotamos por $c \leftarrow \mathcal{E}_k(m) = \mathcal{E}(k, m)$.
3. El algoritmo de descifrado $\mathcal{D}$ actúa sobre el espacio de textos cifrados $\mathcal{C}$, y es determinista. Con una clave $k \in \mathcal{K}$ y un texto cifrado $c \in \mathcal{C}$, devuelve un texto claro $m \in \mathcal{M} \cup \{\perp\}$. Usaremos la notación $m \leftarrow \mathcal{D}_k(c) = \mathcal{D}(k, c)$.

Se dice que el sistema aporta un descifrado correcto si $\forall k \in \mathcal{K}, \quad \forall m \in \mathcal{M}$:

$$\mathbf{Pr}[c \leftarrow \mathcal{E}_k(m) : \quad \mathcal{D}_k(c) = m] = 1$$

Observación 4.1.2 El símbolo $\perp$ anteriormente utilizado se emplea para denotar la salida del algoritmo en el caso de que haya ciertas restricciones sobre los espacios de mensajes y de cifrado, como fuera por ejemplo solo los mensajes de longitud múltiplo de n o haber establecido una longitud máxima.

Ejemplo 4.1.2 El One Time Pad. El *One Time Pad* es un ejemplo sencillo de un sistema de clave privada. Para su construcción fijamos un $l \in \mathbf{N}$. Entonces tenemos que el espacio de claves, mensajes y textos cifrados son $\mathcal{K} = \mathcal{M} = \mathcal{C} = \{0,1\}^l$, es decir, cadenas de l bits.

El algoritmo generador de claves $\mathcal{G}$ genera claves uniformes $k \in \mathcal{K} = \{0,1\}^l$, de forma que todos los elementos de $\{0,1\}^l$ tienen la misma probabilidad de salir.

El algoritmo de cifrado $\mathcal{E}$ recibe un mensaje $m \in \mathcal{M}$ y una clave $k \in \mathcal{K}$ y hace $c = \mathcal{E}(k, m) = k \oplus m$. Donde $\oplus$ es la suma de $\mathbf{Z}_2$, que se puede ver como una operación XOR entre bits.

El algoritmo de descifrado $\mathcal{D}$ recibe un texto cifrado $c \in \mathcal{C}$ y una clave $k \in \mathcal{K}$ y hace $\mathcal{D}(k, c) = k \oplus c = k \oplus (k \oplus m) = (k \oplus k) \oplus m = m$.

Por ejemplo, si queremos enviar $m = 0111$ y tenemos la clave 0110, hacemos $c_i = m_i \oplus k_i$ para cada bit $i \in \{1, 2, 3, 4\}$. Por lo que nos queda $c = 1110$

Para descifrar, realizamos la misma operación. $m_i = c_i \oplus k_i$ para cada $i \in \{1,2,3,4\}$. Entonces tenemos de vuelta que $m = 0111$.

Observación 4.1.3 El cifrado es determinista, y por tanto, la probabilidad de que el descifrado sea correcto es 1.

Definición 4.1.4 Seguridad Incondicional. Decimos que un esquema de cifrado simétrico $\mathcal{SE} = (\mathcal{G}, \mathcal{E}, \mathcal{D})$ con textos claros en un conjunto finito M, textos cifrados en un conjunto finito C y claves en un conjunto finito K, es *incondicionalmente seguro* o *seguro en el sentido de teoría de la información* si se cumple que para toda distribución de probabilidad ϕ definida sobre $\mathcal{M}$[1], $\forall m \in \mathcal{M}, \forall c \in \mathcal{C}$ con $\Pr[C = c] > 0$:

$$\Pr[m \xleftarrow{\phi} M | c \xleftarrow{\phi} C] = \Pr[m \xleftarrow{\phi} M],$$

donde $\xleftarrow{\phi}$ denota selección con respecto a la distribución ϕ.

Informalmente, esta definición captura el hecho de que un texto cifrado c no aporta información alguna para el adversario sobre el texto claro que enmascara.

De manera equivalente, la definición anterior puede escribirse imponiendo:

$$\forall m_0, m_1 \in \mathcal{M}, \forall c \in \mathcal{C} \quad \Pr[\mathcal{E}_K(m_0) = c] = \Pr[\mathcal{E}_K(m_1) = c]$$

Proposición 4.1.5 *El One Time Pad es seguro incondicionalmente si las claves empleadas son de un sólo uso.*

Demostración:
Sea una distribución ϕ sobre $\mathcal{M}$, dados $m \in \mathcal{M}$ y $c \in \mathcal{C}$ cualesquiera, tenemos que para el One Time Pad:

$$\Pr[c \xleftarrow{\phi} C | m \xleftarrow{\phi} M] = \Pr[c \xleftarrow{\phi} M \oplus K | m \xleftarrow{\phi} M] = \Pr[c \xleftarrow{\phi} m \oplus K] = \Pr[m \oplus c \xleftarrow{\phi} K] = \frac{1}{2^l}$$

Por tanto, para cualquier $m_0, m_1 \in \mathcal{M}, \forall c \in \mathcal{C}$:

$$\Pr[c \xleftarrow{\phi} C | m_0 \xleftarrow{\phi} M] = \frac{1}{2^l} = \Pr[c \xleftarrow{\phi} C | m_1 \xleftarrow{\phi} M]$$

Por tanto, el One Time Pad es seguro. En el caso de repetir alguna clave, por ejemplo si los mensajes m_0 y m_1 son cifrados bajo la misma clave k. El adversario puede obtener $c_0 = m_0 \oplus k$ y $c_1 = m_1 \oplus k$, y por tanto obtener la información $c_0 \oplus c_1 = (m_0 \oplus k) \oplus (m_1 \oplus k) = m_0 \oplus m_1$, comprometiendo la comunicación. □

[1] Nótese que también ϕ define entonces una distribución sobre C. Análogamente, escribiremos $c \leftarrow C$ denotando la selección de mensajes en C que es consecuencia de cifrar mensajes claros elegidos con ϕ.

En la actualidad, los cifrados de clave privada difieren bastante del One Time Pad. De entre los algoritmos más destacados tenemos DES (Data Encryption Standard) y AES (Advanced Encryption Standard), que hacen uso de cifrado en bloques, dividiendo el mensaje en distintos bloques de tamaño fijo, y empleando distintas claves derivadas de la clave original. En DES se aplican sucesivos cifrados sobre los distintos bloques, utilizando las salidas anteriores como entradas de los siguientes y realizando también permutaciones sobre los mismos; mientras que en AES se hace uso de matrices y permutaciones entre sus filas y columnas.

Estos algoritmos se basan en el uso de distintas claves derivadas de una clave privada, así como la aplicación de técnicas iterativas, repitiendo ciertas operaciones y procedimientos en sus algoritmos. El objetivo final que se busca con esto es modificar el texto llevando a cabo procesos de confusión y difusión, en los cuales se llevan a cabo sustituciones para enmascarar la información, y se realizan permutaciones para romper los patrones lineales y evitar así reversibilidad.

4.1.3. Sistemas de clave pública

En los sistemas de clave pública o asimétrica intervienen 2 claves: una pública y otra privada. Estas claves están relacionadas entre sí, de forma que para cifrar un mensaje, se utiliza la clave pública del receptor, que lo descifra con su clave privada, sólo conocida por él. Por tanto, el receptor es el único que puede descifrar los mensajes que le mandan. Puede asumirse que este par de claves es generado o certificado por un tercero del que se fíen todas las partes (como una entidad confiable), así como que todas las claves públicas serán accesibles a todos los usuarios de un cierto sistema criptográfico.

La información pública en estos sistemas consiste, de nuevo, en la estructura de todos los algoritmos involucrados (incluyendo los conjuntos donde toman sus entradas y caen sus salidas) y las claves públicas. Por otro lado, las claves privadas permanecen en secreto y no se comparten con nadie.

Con los sistemas de clave privada, surgía un gran problema a la hora de distribuir la clave de forma segura. Es decir, ambas partes debían conocer la clave, y ésta a la vez debía permanecer en secreto, lo que plantea una serie de dificultades. Con los sistemas de clave pública, este problema se soluciona, pues ahora cada una de las partes elige su clave privada, sin tener que comunicársela ni compartirla con nadie, y posteriormente, a partir de ella, genera una clave pública que será accesible y comunicada a todo el mundo públicamente.

Si denotamos pk a las clave pública y sk a la privadas, la descripción informal de un cifrado de clave pública es como sigue:

$$\begin{array}{ccc} \mathcal{A} & \longrightarrow & \mathcal{B} \\ m & & \\ c = \mathcal{E}(pk_B, m) & \longrightarrow & c \\ & & m = \mathcal{D}(sk_B, c) \end{array}$$

Definición 4.1.6 Sistema de clave pública. Un *sistema de clave pública* se define como una terna de algoritmos probabilísticos y polinomiales $(\mathcal{G}, \mathcal{E}, \mathcal{D})$, que satisfacen [2]:

1. $\mathcal{G}$ es el algoritmo de generación de claves que produce un par (pk, sk), donde pk denominada clave pública, y sk es la clave privada, a partir de un parámetro de seguridad 1^n. Lo escribimos como $(pk, sk) \leftarrow \mathcal{G}(1^n)$ o $(pk, sk) \in \mathcal{G}(1^n)$.
2. $\mathcal{E}$ es el algoritmo de cifrado, que toma como entradas el parámetro de seguridad 1^n, una clave pública pk obtenida del algoritmo de generación $\mathcal{G}$ y una cadena de texto claro $m \in \mathcal{M}$ que llamamos mensaje, y produce como salida una cadena $c \in \mathcal{C}$ que es el texto cifrado. Lo denotamos como $c \in \mathcal{E}(1^n, pk, m)$, o como $c \leftarrow \mathcal{E}_{pk}(m) = \mathcal{E}(pk, c)$.
3. $\mathcal{D}$ es el algoritmo de descifrado, que toma como entrada una clave privada sk obtenida del algoritmo de generación $\mathcal{G}$ y el texto cifrado $c \in \mathcal{E}(1^n, pk, m)$ y devuelve como salida una cadena $m' \in \mathcal{M}$. También lo podemos denotar como $m \leftarrow \mathcal{D}_{sk}(c) = \mathcal{D}(sk, c)$.

Se dice que el sistema aporta un descifrado correcto si $\forall (pk, sk) \in \mathcal{G}(1^n), \quad \forall m \in \mathcal{M}$:

$$\mathbf{Pr}[c \leftarrow \mathcal{E}(1^n, pk, m) : \quad \mathcal{D}(1^n, sk, c) \neq m] \text{ es despreciable.}$$

Ejemplo 4.1.4 Criptosistema de Rabin. El *criptosistema de Rabin* descrito en [8] es un sistema de cifrado de clave pública que ilustra bien cómo funcionan los sistemas asimétricos.

En cuanto al algoritmo de generación de clave $\mathcal{G}$, si tomamos n como parámetro de seguridad, se genera de salida: la clave privada como una tupla formada por dos números primos de n bits p, q tales que $p \equiv 3 \bmod 4$ y $q \equiv 3 \bmod 4$, y la clave pública, que es el producto de los dos anteriores, $N = p \cdot q$. Por tanto $(pk, sk) = (N, (p, q)) \in \mathcal{G}(1^n)$.

Para proceder al cifrado de un texto claro m, el algoritmo de cifrado $\mathcal{E}$ computa la operación $c \equiv m^2 \bmod N$. Y para descifrar c, el algoritmo de descifrado $\mathcal{D}$ sigue los siguientes pasos, como se explica en [8]:

1. Calculamos m_p y m_q tales que:
   a) $m_p \equiv c^{\frac{1}{4}(p+1)} \bmod p$
   b) $m_q \equiv c^{\frac{1}{4}(pq+1)} \bmod q$
2. Usando el algoritmo de Euclides para encontrar y_p y y_q de forma que $mcd(p, q) = p \cdot y_p + q \cdot y_q = 1$.
3. Usando el Teorema del Resto Chino, podemos deducir que m es una de las siguientes 4 opciones:
   a) $m_1 \equiv (p \cdot y_p \cdot m_q + q \cdot y_q \cdot m_p) \bmod N$
   b) $m_2 \equiv n - r_1$
   c) $m_3 \equiv (p \cdot y_p \cdot m_q - q \cdot y_q \cdot m_p) \bmod N$

d) $m_4 \equiv n - r_3$

A priori no tendríamos forma de saber cuál es el i tal que $m_i = m$ sin más información. Una opción sería tener dos pares de claves diferentes y enviar un mismo mensaje de las dos formas, y de esta manera en la mayoría de ocasiones podríamos recuperar el mensaje original.

Por ejemplo, si tenemos $p = 3, q = 7 \longrightarrow N = 3 \cdot 7 = 21$. Si queremos enviar $m = 10$, entonces $c = 10^2 \bmod 21 \longrightarrow c = 16$.

Para descifrar seguimos los pasos anteriores:

1. Calculamos m_p y m_q:

   a) $m_p = 16^{\frac{1}{4}(4) \bmod 3} = 16 \bmod 3 = 1$

   b) $m_q = 16^{\frac{1}{4}(12) \bmod 7} = 32 \bmod 7 = 4$

2. Aplicamos Euclides extendido:

$$7 = 3 \cdot 2 + 1 \longrightarrow 1 = 7 - 3 \cdot 2 \longrightarrow y_p = -2, y_q = 1$$

3. Computamos las posibles raíces:

   a) $r_1 = 4$

   b) $r_2 = 17$

   c) $r_3 = 11$

   d) $r_4 = 10$

Como podemos ver, $r_4 = m = 10$, pero sólo con esta información no podemos deducirlo completamente.

Si además ciframos m con otra clave, $(p = 11, q = 7), N = 77$, entonces $c = 23$. Y si repetimos todo el proceso anterior, obtenemos:

1. $m_p = 1$, $m_q = 4$

2. $y_p = 1$, $y_q = 0$

3. $r_1 = 67$, $r_2 = 10$, $r_3 = 32$, $r_4 = 45$

Como vemos, el único que coincide es $m = 10$, que es el mensaje original.

4.2. Análisis formal seguridad para esquemas de cifrado de clave privada

Tal y como se reflexiona en [4], uno de los principales problemas en criptografía es definir qué es la seguridad. Muchas veces, es mejor pensar en términos de inseguridad, ya que las nociones son mas intuitivas. Por ejemplo, para el caso de cifrado es evidente que si se puede recuperar la clave secreta, o se puede recuperar el texto claro completo a partir de uno cifrado, el esquema no es seguro. Pero es un error pensar que porque no se pueda obtener la clave o no se pueda recuperar todo el mensaje completo, el esquema es seguro. Esto se debe a que, aunque no se pueda recuperar todo el mensaje, se puede llegar a obtener información valiosa sobre él. Ésta información puede ser algún bit, la suma de los bits del mensaje, etc. que puede ser información que no queremos que sea comprometida. Por tanto, un enfoque habitual es tomar como definición de seguridad la exigencia de que los adversarios no sean capaces de obtener ninguna información a partir de los textos cifrados.

En esta sección vamos a introducir dos conceptos sobre seguridad en cifrados: la *indistinguibilidad o seguridad semántica* y la *maleabilidad*. Distinguiremos entre dos tipos de ataques que se pueden realizar sobre un sistema: los ataques de indistinguibilidad sobre textos planos (Chosen Plaintext Attack – CPA), conocidos como *ataques pasivos* y los ataques sobre textos cifrados escogidos (Chosen Ciphertext Attack – CCA), llamados también *ataques activos*.

En este documento nos centraremos en los ataques sobre esquemas de clave privada, ya que el esquema escogido para analizar es de este tipo, pero de forma análoga hay una teoría similar que aplica a los esquemas de clave pública.

Indistinguibilidad. Un concepto que está muy relacionado con la forma anterior de definir la seguridad de un esquema de cifrado es la *indistinguibilidad* (también llamada seguridad semántica). De forma intuitiva e informal, esta propiedad representa la capacidad de un sistema de aportar o no cierta información al adversario acerca de los cifrados que se intercambian por el canal, sin tener por qué saber nada de la clave o el texto claro completo. Se dice que un esquema es seguro en el sentido de la indistinguibilidad, si dados dos mensajes m_0, m_1 y el cifrado asociado a uno de ellos, un adversario puede acertar con una probabilidad mayor de $0,5$ (que es la de acertar aleatoriamente, y no depende del esquema) y en tiempo polinomial cuál de ellos es el que ha sido cifrado.

Maleabilidad. Informalmente, diremos que un esquema es *maleable* si, dados un texto cifrado c asociado a un mensaje m, es posible construir otro texto cifrado c' que es el resultado de cifrar un m', que a su vez está relacionado de una manera conocida con m. Es decir, si podemos construir un cifrado c' de, por ejemplo, $m' = 2m$ o $m' = m - 1$ sin la necesidad de conocer m. Esta idea se generaliza fácilmente al caso de tener varios textos cifrados.

Antes de entrar a explicar los ataques CPA y CCA, recuperamos el concepto de Parámetro de Seguridad (1^n) introducido anteriormente en la definición y explicación de los sistemas

de clave pública, donde no se ha profundizado mucho en el concepto.

Parámetro de seguridad. En [4] el *parámetro de seguridad* se define como un número entero que afecta tanto a la seguridad del esquema como la de las partes implicadas, así como al atacante. Es conocido por el adversario, y se toman el tiempo de ejecución y la probabilidad de éxito del mismo como funciones de n. Consideraremos adversarios que, en principio, son solo capaces de ejecutar algoritmos probabilísticos en tiempo polinomial, es decir, diremos que hay un ataque eficiente si existe un polinomio P para el que el adversario es capaz de ejecutar su ataque al menos en $P(n)$ tiempo.

Por tanto, cuanto mayor sea el parámetro de seguridad, en principio lo razonable es que el adversario tarde más tiempo en romper el cifrado. En los ejemplos vistos de clave pública se tomaba como parámetro de seguridad el número de bits de las claves, ya que cuanto más grandes y largos sean estos números, habrá mas dificultad en factorizarlos y, por tanto, será más difícil romper los cifrados.

Hay que tener en cuenta que al aumentar el parámetro de seguridad, también se aumenta el volumen de cómputo de los algoritmos y, por ende, el tiempo de ejecución de los mismos. Es por esto que hay que encontrar un punto medio entre velocidad de ejecución y seguridad de dichos algoritmos, fijando un parámetro de seguridad que, usualmente, se establece en el mínimo para poder ejecutarlos en el menor tiempo posible, y a la vez poder proveer de un mínimo de seguridad ante ataques.

En los sistemas de clave privada, a diferencia de los sistemas de clave pública, no se toma el parámetro de seguridad como una entrada del algoritmo de generación de clave, sino que se toma como un parámetro fijo en el sistema.

Hablemos ahora de la noción de *oráculo*, necesaria en el análisis de seguridad posterior.

Oráculo. Un oráculo $\mathcal{O}_{\mathcal{A}}$ es un componente que ejecuta un algoritmo $\mathcal{A}$ de forma transparente al que lo ejecuta. Es decir, el usuario de $\mathcal{O}$ no conoce toda la información cómo funciona $\mathcal{A}$ (por ejemplo, no conoce la clave privada), pero puede usarlo. Usaremos oráculos a continuación para los experimentos sobre ataques CPA y CCA, donde el adversario tendrá acceso a distintos algoritmos de cifrado y descifrado, pero no podrá ver cómo funcionan.

Por último, hablemos brevemente del concepto de *función despreciable*.

Función despreciable. Para las próximas definiciones, haremos uso del concepto de función despreciable. Intuitivamente, una función despreciable es aquella que crece asintóticamente más despacio que cualquier polinomio positivo. Es decir, suponiendo que tenemos un algoritmo que corre en un tiempo representado por una función $\epsilon(n)$, ésta es una *función despreciable* si, repitiendo dicho algoritmo un número polinómico de veces $P(n)$ tenemos que $\epsilon(n) \cdot P(n)$ sigue siendo una función despreciable. Formalmente, una función despreciable se define como sigue.

Definición 4.2.1 Función despreciable ϵ Se dice que la función $\epsilon : \mathbb{N} \longrightarrow \mathbb{R}^+$ es

despreciable si para todo polinomio positivo $P(x)$ existe un $N \in \mathbb{N}$ tal que $\forall n \in \mathbb{N}, n > N$ se tiene que:

$$\epsilon(n) < \frac{1}{P(n)}$$

4.2.1. Ataques CPA (indistinguibilidad)

La idea detrás de los ataques sobre textos planos es la siguiente: el atacante puede cifrar una serie de mensajes $m_1, m_2, \ldots$ a través de un oráculo $\mathcal{O}_\mathcal{E}$ (sin conocer la clave privada), y observa los cifrados $c_1, c_2, \ldots$. Después de esto, elige dos mensajes m_0, m_1 y se cifra uno de ellos elegido uniformemente al azar, obteniendo c como resultado.

Informalmente, diremos que un esquema es seguro contra ataques CPA si el adversario no es capaz de saber si c ha sido el cifrado de m_0 o m_1 con más probabilidad que la de elegir cuál a sido el caso de forma aleatoria.

Como ya hemos ido explicando, los ataques sobre textos planos elegidos tratan de un adversario que puede cifrar varios mensajes $m_1, m_2, \ldots$ a través de un oráculo de cifrado $\mathcal{O}_\mathcal{E}$, es decir, sin conocer la clave privada. De este modo, un esquema es seguro contra ataques CPA si tras obtener las salidas del oráculo, el adversario que obtiene un mensaje c en el canal (diferente de los cifrados que obtuvo del oráculo) no es capaz de identificar si se corresponde con un cierto texto claro; de hecho, no obtiene ninguna información sobre el texto claro que subyace a c. De modo más formal:

Definición 4.2.2 Experimento de indistinguibilidad en ataques CPA. Consideramos el esquema de clave privada $\Pi = (\mathcal{G}, \mathcal{E}, \mathcal{D})$, y denotando $\mathcal{A}$ al adversario, y un parámetro de seguridad n, entonces el *experimento de indistinguibilidad en ataques CPA sobre* Π, $PrivK^{cpa}_{\mathcal{A},\Pi}(n)$ se define en [4] como:

1. Se genera una clave k ejecutando $\mathcal{G}(1^n)$
2. El adversario recibe de entrada 1^n y acceso al Oráculo de $\mathcal{O}_\mathcal{E}$, y produce como salida un par de mensajes m_0, m_1 de la misma longitud.
3. Se elige un bit $b \in \{0, 1\}$ y se le envía el texto cifrado $c \leftarrow \mathcal{O}_\mathcal{E}(m_b)$ a $\mathcal{A}$
4. $\mathcal{A}$ sigue teniendo acceso al Oráculo de $\mathcal{O}_\mathcal{E}$ y produce como salida un bit b'.
5. La salida del experimento es 1 si $b = b'$, 0 en caso contrario. Si ocurre lo primero, se dice que $\mathcal{A}$ ha tenido éxito.

Se dice que un esquema de clave privada $\Pi = (\mathcal{G}, \mathcal{E}, \mathcal{D})$ tiene cifrados indistinguibles bajo ataques de elección de textos planos, o que es seguro bajo ataques CPA si para todo adversario probabilístico de tiempo polinomial $\mathcal{A}$ existe una función despreciable ϵ tal que:

$$\mathbf{Pr}[\mathrm{PrivK}^{\mathrm{cpa}}_{\mathcal{A},\Pi} = 1] \leq \frac{1}{2} + \epsilon(n)$$

4.2.2. Ataques CCA (indistinguisibilidad)

En los ataques de textos cifrados elegidos tenemos la misma idea que en CPA, pero esta vez el adversario tiene, además de el Oráculo de cifrado $\mathcal{O}_{\mathcal{E}}$, el de descifrado $\mathcal{O}_{\mathcal{D}}$.

Definición 4.2.3 Experimento de indistinguibilidad en ataques CCA. Consideramos el esquema de clave privada $\Pi = (\mathcal{G}, \mathcal{E}, \mathcal{D})$, y denotando $\mathcal{A}$ al adversario, y un parámetro de seguridad n, entonces el *experimento de indistinguibilidad en ataques CCA sobre* Π, $PrivK^{cca}_{\mathcal{A},\Pi}(n)$ se define en [4] como:

1. Se genera una clave k ejecutando $\mathcal{G}(1^n)$
2. El adversario recibe de entrada 1^n y acceso a los Oráculos de $\mathcal{O}_{\mathcal{E}}$ y $\mathcal{O}_{\mathcal{D}}$, y produce como salida un par de mensajes m_0, m_1 de la misma longitud.
3. Se elige un bit $b \in \{0, 1\}$ y se le envía el texto cifrado $c \leftarrow \mathcal{O}_{\mathcal{E}}(m_b)$ a $\mathcal{A}$
4. $\mathcal{A}$ sigue teniendo acceso a los Oráculos de $\mathcal{O}_{\mathcal{E}}$ y $\mathcal{O}_{\mathcal{D}}$, pero no puede introducir el texto cifrado c del desafío, y finalmente produce como salida un bit b'.
5. La salida del experimento es 1 si $b = b'$, 0 en caso contrario. Si ocurre lo primero, se dice que $\mathcal{A}$ ha tenido éxito.

Se dice que un esquema de clave privada $\Pi = (\mathcal{G}, \mathcal{E}, \mathcal{D})$ tiene cifrados indistinguibles bajo ataques de elección de textos cifrados, o que es seguro bajo ataques CCA si para todo los adversario probabilístico de tiempo polinomial $\mathcal{A}$ existe una función despreciable ϵ tal que:

$$\mathbf{Pr}[\mathrm{PrivK}^{\mathrm{cca}}_{\mathcal{A},\Pi} = 1] \leq \frac{1}{2} + \epsilon(n).$$

Capítulo 5

Esquema de Koscienly et al.

5.1. Introducción y contexto.

En este capitulo vamos a describir y explicar con detalle el esquema presentado en [6], que a su vez se basa en los estudios descritos en [5], y será la base con la que vamos a estar trabajando durante el resto del documento.

Aunque en la propuesta original se hace referencia a que el sistema presentado es de clave pública, durante todo el protocolo se emplea una única clave secreta, tanto para cifrar como para descifrar. Por tanto este esquema, cuyo funcionamiento se basa en la estructura y propiedades de los cuasigrupos, es realmente de clave privada.

Vamos a utilizar las notaciones que se han venido presentando a lo largo de las distintas secciones anteriores, aunque no sean las mismas que usan los autores en el texto original, para mantener continuidad y coherencia en todo el texto, y que así sea más fácil de entender.

5.2. Presentación del esquema.

Dado un cuasigrupo $Q = (\mathcal{A}, \oplus)$, con $\#\mathcal{A} = q \geq 3$, dotado de una operación interna $\oplus$, fijado un $n \in \mathbb{N}$ se definen el espacio de mensajes, claves y cifrados como $\mathcal{A}^n$. Es decir, $\mathcal{C} = \mathcal{M} = \mathcal{A}^n$

En este esquema, la información pública es el cuasigrupo sobre el que se está trabajando, y por tanto las posibles funciones de cifrado y descifrado, que se verán a continuación. Lo que reside de forma privada es tanto la clave, como el par de funciones concreto que se usa para cifrar (toda esa información debe, por tanto, considerarse parte de la clave secreta).

5.2.1. Algoritmo de generación de claves $\mathcal{G}$

En el esquema que nos ocupa, la generación de claves es aleatoria, y se asume que el algoritmo $\mathcal{G}$ genera valores uniformes del espacio de claves $\mathcal{M} = \mathcal{A}^n$.

Este esquema tiene un algoritmo de generación de clave especial, ya que además de la clave secreta k que será utilizada en la comunicación, ofrece como salida dos funciones que serán las empleadas por los algoritmos de cifrado y descifrado. Siendo así, los algoritmos $\mathcal{E}$ y $\mathcal{D}$ dependerán en esencia de la salida de $\mathcal{G}$.

5.2.2. Algoritmos de cifrado y descifrado.

Se definen los algoritmos de cifrado $\mathcal{E}$ y descifrado $\mathcal{D}$ como dos funciones, cuyas entradas son: la clave privada k y, en el caso de $\mathcal{E}$ el mensaje m a cifrar; y en el caso de $\mathcal{D}$ el texto cifrado c a descifrar. De esta forma $\forall i = 1 \dots n$ se cumple:

$$\text{i)} \qquad c_i = \mathcal{E}(m_i, k_i)$$

$$\text{ii)} \qquad m_i = \mathcal{D}(c_i, k_i)$$

$$\text{iii)} \quad m_i = \mathcal{E}(\mathcal{D}(m_i, k_i), k_i).$$

Donde cada k_i, m_i, c_i representan el elemento i-ésimo de la clave, el texto claro y el texto cifrado respectivamente.

Para hacer explícitos los procesos anteriores, utilizamos la operación interna del cuasigrupo, y a partir de ella se derivan dos funciones más, que nos aportan una invertibilidad que no tiene por qué satisfacer el cuasigrupo inicialmente, y se definen como sigue.

Definición 5.2.1 Sea $S(x, y) = x \oplus y$ (luego, también $S(y, x) = y \oplus x$). Se definen las operaciones binarias $\ominus$ y $\oslash$ a través de las igualdades:

$$1. \quad S(x, y) \ominus y = x$$

$$2. \quad S(y, x) \oslash y = x.$$

Definición 5.2.2 Análogamente a lo anterior, se definen las funciones D y $\hat{D}$ de la siguiente forma:

$$\text{i)} \quad D(x, y) = x \ominus y$$

$$\text{ii)} \quad \hat{D}(x, y) = x \oslash y.$$

Observación 5.2.3 Tenemos que aplicando la definición 5.2.1 se cumple:

$$1. \quad D(S(x,y),y) = x$$

$$2. \quad \hat{D}(S(y,x),y) = x.$$

Proposición 5.2.4 *En las condiciones anteriores se tiene:*

$$i) \quad D(x,y) = z \iff S(z,y) = x$$

$$ii) \quad \hat{D}(x,y) = z \iff S(y,z) = x$$

$$iii) \quad D(x,y) = z \iff \hat{D}(x,z) = y.$$

Demostración:
La demostración es trivial.

Las dobles implicaciones $i)$ y $ii)$ se deducen directamente aplicando la proposición 5.2.3, sustituyendo.

La $iii)$ se obtiene de las dos anteriores.

Proposición 5.2.5 ***Propiedades de estas funciones.*** *Sean S, D y $\hat{D}$ bajo las condiciones anteriores. Se siguen las siguientes propiedades:*

$$\left.\begin{array}{ll} 1. & D(S(x,y),y) = x \\ 2. & \hat{D}(S(y,x),y) = x \end{array}\right\} \text{ Observación 5.2.3}$$

$$3. \quad S(D(x,y),y) = x$$

$$4. \quad \hat{D}(y,D(y,x)) = x$$

$$5. \quad D(y,\hat{D}(y,x)) = x$$

$$6. \quad S(y,\hat{D}(x,y)) = x.$$

Demostración:
Para las propiedades de la 3 a la 6 usamos la proposición 5.2.4:

3. $S(\underbrace{D(x,y)}_{z}, y) = S(z,y) \overset{\text{i)}}{=} x$

4. $\hat{D}(y, \underbrace{D(y,x)}_{z}) = \hat{D}(y,z) \overset{\text{iii)}}{=} x$

5. $D(y, \underbrace{\hat{D}(y,x)}_{z}) = D(y,z) \overset{\text{iii)}}{=} x$

6. $S(y, \underbrace{\hat{D}(x,y)}_{z}) = S(y,z) \overset{\text{ii)}}{=} x \quad \square$

Ejemplo 5.2.3 Tabla de D en $\mathbb{Z}_3$. Para poder visualizar mejor cómo se forman D y $\hat{D}$ a partir de S, vamos a hacer un ejemplo en $\mathbb{Z}_3$ con la suma, es decir $Q = (\mathbb{Z}_3, +)$:

Tenemos que la tabla de S, es la tabla de la suma típica en $\mathbb{Z}_3$:

$\oplus$	0	1	2
0	0	1	2
1	1	2	0
2	2	0	1

Ahora calcularíamos la tabla de D a partir de la proposición 5.2.4 y la tabla de S.

Entonces:

1. $D(0,0) = z \iff S(z,0) = 0 \Rightarrow \underline{z = 0}$

2. $D(0,1) = z \iff S(z,1) = 0 \Rightarrow \underline{z = 2}$

3. $D(0,2) = z \iff S(z,2) = 0 \Rightarrow \underline{z = 1}$.

Por tanto, para poder calcular $D(x,y)$, primero necesitamos encontrar un z en la tabla de S para el cual se cumpla que $S(z,y) = x$, y entonces $D(x,y) = z$. De igual forma pasaría con $\hat{D}$.

Y repitiendo esto con las demás filas, terminaríamos la tabla:

$\ominus$	0	1	2
0	0	2	1
1	1	0	2
2	2	1	0

En este caso, al ser $\mathbb{Z}_3$ un grupo abeliano, la suma es conmutativa, y por tanto $S(x,y) = S(y,x) \Rightarrow D = \hat{D}$.

En el caso de tener un cuasigrupo no conmutativo, la tabla de $\hat{D}$ se calcularía de forma análoga usando ii) con la tabla de S o iii) con la de D.

5.2.4. Posibles algoritmos $\mathcal{E}$ y $\mathcal{D}$.

Observando la proposición 5.2.5, vemos que tenemos 6 posibles pares de funciones para ser utilizadas en los algoritmos $\mathcal{E}$ y $\mathcal{D}$.

$$(\mathcal{E}, \mathcal{D}) \in \{(S, D), (S, \hat{D}), (D, \hat{D}), (D, S), (\hat{D}, S), (\hat{D}, D)\}.$$

Se destacan una serie de hechos sobre esto:

Observación 5.2.6 Ya se ha mencionado en el ejemplo con $\mathbb{Z}_3$ que si el cuasigrupo en cuestión tiene conmutatividad, entonces D = $\hat{D}$ y, por tanto, se reducen las parejas de funciones posibles a 3, que serían (S, D),(D, S), o (D, D).

Observación 5.2.7 Notar que dependiendo del par de funciones escogidas, el orden de k y m (o en el caso de descifrado, c) como argumentos de las funciones puede variar.

Es decir, puede ser que empleando una misma función para el cifrado, el resultado que produzca sea diferente. Esto puede ocurrir si la operación no es conmutativa. Por ejemplo si se emplea el par $(D, \hat{D})$, tenemos $c_i = D(k_i, m_i)$, pero si usamos (D, S) tenemos $c_i = D(m_i, k_i)$. De igual forma ocurre en el descifrado. Para descifrar en este último caso hacemos $m_i = S(c_i, k_i)$, pero si usáramos $(\hat{D}, S)$ descifraríamos de esta otra manera $m_i = S(k_i, c_i)$.

Esto se debe a la proposición 5.2.5, puede comprobarse asignando $x = m_i$ y $y = k_i$, para cada i.

Capítulo 6

Implementación: ejemplos concretos.

Vamos a introducir una serie de familias de cuasigrupos, e ilustrar cómo se produciría el proceso completo de cifrado y descifrado de los mensajes a modo de distintos ejemplos de implementación. En estos escenarios, utilizaremos los mismos cuasigrupos que hemos venido introduciendo durante todo el trabajo y con los que nos hemos familiarizado para realizar el criptoanálisis.

Notación. A partir de ahora vamos a emplear un abuso de notación. Usaremos las funciones de cifrado y descifrado, así como las operaciones asociadas a los cuasigrupos sobre los espacios de mensajes, cifrados y claves como sigue. Tomando como ejemplo la función S o el operador $\oplus$, dados un mensaje m y una clave k, denotamos $c = S(m,k) = m \oplus k$ al resultado de operar m y k elemento a elemento. Es decir, $c_i = S(m_i, k_i) = m_i \oplus k_i, \quad \forall i \in \{1 \dots n\}$.

6.1. Ejemplo en un grupo abeliano.

Aprovechando que hemos calculado las tablas de S y D de $Q = (\mathbb{Z}_3, \oplus)$, detallamos el esquema objetivo sobre este mismo grupo.

Supongamos que el algoritmo $\mathcal{G}$ nos genera una clave $k = 0221$, así como que el par de funciones que usaremos es (D, S), es decir, ciframos con D y desciframos con S. Y queremos enviar el mensaje $m = 1102$.

Cabe recordar que, en el caso abeliano, tenemos únicamente 3 posibles pares de funciones: (S, D),(D, S), y (D, D), ya que $D = \hat{D}$ por ser S conmutativa.

Entonces para calcular el texto cifrado, hacemos $c = \mathcal{E}(m,k) = D(m,k) = D(1102, 0221) = 1211$, ya que $D(1,0) = 1$, $D(1,2) = 2$, $D(0,2) = 1$ y $D(2,1) = 1$.

De igual forma, el receptor recupera m con $\mathcal{D}$ haciendo $m = \mathcal{D}(c,k) = S(c,k) = S(1211, 0221) =$

1102.

6.2. Ejemplo en un grupo de permutaciones.

Podemos tomar el grupo de permutaciones $(S_3, \circ)$, que se trata de todas las permutaciones posibles sobre $\mathbb{Z}_3$, por lo que es un conjunto con 6 elementos $S_3 = \{id, f, g, h, i, j\}$.

Recordemos cuáles son estas permutaciones para poder observar las tablas de S, D y $\hat{D}$ que se generan:

$$id = (\quad) \qquad f = (0 \quad 1 \quad 2) \qquad g = (0 \quad 2 \quad 1)$$

$$h = (1 \quad 2) \qquad i = (0 \quad 2) \qquad j = (0 \quad 1)$$

Se puede comprobar que la tabla de la operación de la composición con estas definiciones queda así:

$\circ$	id	f	g	h	i	j
id	id	f	g	h	i	j
f	f	g	id	j	h	i
g	g	id	f	i	j	h
h	h	i	j	id	f	g
i	i	j	h	g	id	f
j	j	h	i	f	g	id

A partir de esta tabla de la función S y utilizando la proposición 5.2.4, se pueden derivar las tablas de D y $\hat{D}$:

D	id	f	g	h	i	j
id	id	g	f	h	i	j
f	f	id	g	j	h	i
g	g	f	id	i	j	h
h	h	j	i	id	f	g
i	i	h	j	g	id	f
j	j	i	h	f	g	id

$\hat{D}$	id	f	g	h	i	j
id	id	g	f	h	i	j
f	f	id	g	i	j	h
g	g	f	id	j	h	i
h	h	i	j	id	g	f
i	i	j	h	f	id	g
j	j	h	i	g	f	id

Ahora veamos el ejemplo en el caso de que el algoritmo $\mathcal{G}$ haya proporcionado como salida la clave $k = fg(id)j$, así como el par de funciones (S, D). Si consideramos el texto claro $m = hihi$.

Entonces ciframos con $\mathcal{E}$, según la operación de composición:

$$c = \mathcal{E}(m,k) = S(m,k) = S(hihi, fg(id)j) = ihhf.$$

Así, recuperamos el mensaje original con $\mathcal{D}$, según la tabla de D:

$$m = \mathcal{D}(c,k) = D(c,k) = D(ihhf, fg(id)j) = hihi.$$

6.3. Ejemplo usando un cuadrado latino.

Como hemos visto, un cuadrado latino también forma la tabla de operación de un cuasigrupo, por lo que vamos a usar un cuasigrupo regido por el siguiente cuadrado latino:

$\oplus$	0	1	2	3
0	3	0	1	2
1	2	3	0	1
2	1	2	3	0
3	0	1	2	3

Curiosamente, cuando vamos a hacer las tablas de D y $\hat{D}$, nos damos cuenta de que $S = D$, y que la tabla de $\hat{D}$ solo intercambia las filas impares de la tabla de S:

$\hat{D}$	0	1	2	3
0	1	2	3	0
1	2	3	0	1
2	3	0	1	2
3	0	1	2	3

Supongamos que el algoritmo $\mathcal{G}$ nos haya generado una clave $k = 3320$, usando el par de funciones $(\hat{D}, S)$. Si queremos enviar el mensaje $m = 1023$.

Entonces: $c = \mathcal{E}(m,k) = \hat{D}(m,k) = \hat{D}(1023, 3320) = 1010$, y desciframos con $m = \mathcal{D}(c,k) = S(k,c) = S(3320, 1010) = 1023$.

6.4. Ejemplo usando un sistema de Steiner

Vamos a incluir también el sistema de Steiner presentado anteriormente en los fundamentos matemáticos $(S, \mathcal{B})$ basado en espacios proyectivos sobre F_2, por lo que el alfabeto en este caso se compondrá de 6 elementos $S = \{1 \dots 7\}$, y donde

$$\mathcal{B} = \{246, 145, 347, 123, 257, 167, 356\}$$

Utilizando la notación $abc \equiv \{a, b, c\}$.

Como ya hemos mencionado, el sistema de Steiner forma cuasigrupo con la operación $\oplus$ cuando se cumple que:

$$x \oplus y = z \iff \{x, y, z\} \in \mathcal{B} \vee x = y = z.$$

Por lo que nos quedaría la siguiente tabla de operación:

S	1	2	3	4	5	6	7
1	1	3	2	5	4	7	6
2	3	2	1	6	7	4	5
3	2	1	3	7	6	5	4
4	5	6	7	4	1	2	3
5	4	7	6	1	5	3	2
6	7	4	5	2	3	6	1
7	6	5	4	3	2	1	7

Además, como tenemos simetría total, se cumple que $S = D = \hat{D}$, por lo que solo existe un posible par de funciones de cifrado y descifrado: (S, S).

Si queremos enviar el mensaje $m = 2461$, usando la clave generada $k = 1234$, entonces usamos S tanto para cifrar como para descifrar: $c = \mathcal{E}(m, k) = S(m, k) = S(2461, 1234) = 3655$, y $m = \mathcal{D}(c, k) = S(c, k) = S(3655, 1234) = 2461$.

6.5. Ejemplo usando un espacio vectorial finito

Por último, vamos a ver un cuasigrupo $(V, \oplus)$ formado por el espacio vectorial V de dimensión 3, que toma los escalares sobre el cuerpo $\mathbb{Z}_5$, y donde:

$$\begin{array}{rccc} \oplus: & V \times V & \longrightarrow & V \\ & (u, v) & \mapsto & u \oplus v = \frac{u+v}{2}. \end{array}$$

Debido a la gran cantidad de elementos, no es factible realizar una tabla de las funciones del cuasigrupo, pero podemos obtener una expresión de las funciones utilizando la proposición 5.3.4. Tenemos que:

$$D(u, v) = w \overset{5,3,4}{\iff} S(w, v) = u \iff \frac{w + v}{2} = u \iff w = 2u - v \iff D(u, v) = 2u - v.$$

Además, como la suma de vectores es conmutativa, tenemos que $D = \hat{D}$.

Ahora sean $u = (0, 1, 3), v = (2, 4, 4), w = (1, 0, 1)$, y queremos enviar el mensaje $m = uv$, usando la clave $k = ww$, bajo las funciones (S, D), generadas por $\mathcal{G}$.

Entonces ciframos con:

$$c = \mathcal{E}(m, k) = S(m, k) = S(uv, ww) =$$

$$= S((0, 1, 3)(2, 4, 4), (1, 0, 1)(1, 0, 1)) = \frac{(1, 1, 4)}{2}\frac{(3, 4, 5)}{2} = (3, 3, 2)(4, 2, 0).$$

Y desciframos con:

$$m = \mathcal{E}(c, k) = D(c, k) = D((3, 3, 2)(4, 2, 0), (1, 0, 1)(1, 0, 1)) =$$

$$= (2(3, 3, 2) - (1, 0, 1))(2(4, 2, 0) - (1, 0, 1)) = uv.$$

Capítulo 7

Criptoanálisis.

Como mencionamos en la introducción del apartado 4, el criptoanálisis es la rama de la criptología que se dedica a estudiar los sistemas criptográficos y buscar posibles errores o fallos en la seguridad de los mismos. En esta sección vamos a llevar a cabo un estudio de la seguridad o criptoanálisis del esquema presentado en las secciones anteriores, presentando y empleando diferentes técnicas. Adicionalmente, se propondrán posibles reimplementaciones, mejoras o soluciones a los distintos fallos encontrados.

7.1. Primer nivel: importancia de la aleatorización

Un factor crucial a la hora de estudiar la seguridad bajo ataques CPA y/o CCA es la aleatorización de los algoritmos de cifrado. La causa de ello es que si el algoritmo no implementa este factor de aleatorización, es decir, es determinista, la seguridad es nula, pues siempre se superarán los experimentos presentados en las secciones 4.2.1 y 4.2.2.

Esto se debe a que el adversario siempre tiene acceso al oráculo de cifrado $\mathcal{E}$, y éste, al cifrar de forma determinista, siempre cifra igual un mismo mensaje.

En nuestro caso, el esquema presentado en [6] y que hemos analizado con detalle en la sección 5 emplea una forma determinista de cifrar, pues aunque con una cierta configuración sobre una determinada estructura algebraica podemos tener a nuestra disposición varios posibles cifrados, una vez escogido uno, éste no cambia y, por tanto, el oráculo cifra un mismo mensaje de la misma forma. De esta manera, haciendo una sencilla comparación de los textos cifrados pasaríamos los experimentos exitosamente.

Para el ataque CPA, si tenemos los mensajes m_0 y m_1 y el texto cifrado del reto c, podemos utilizar el oráculo de cifrado $\mathcal{E}_k$, de forma que, como m_0 y m_1 siempre cifran de la misma forma y tenemos total acceso al oráculo, hacemos $c_0 = \mathcal{E}_k(m_0)$. Si $c = c_0$, entonces $b' = 0$, y si no, $b' = 1$. En cualquier caso habríamos tenido éxito.

Del mismo modo, para un ataque CCA, aunque no podemos introducir c en el oráculo de descifrado $\mathcal{D}_k$, podemos seguir utilizando el mismo método de comparación de los textos cifrados anterior con el oráculo de cifrado $\mathcal{E}_k$.

Por el contrario, si el algoritmo de cifrado contiene un componente aleatorio, el oráculo ya no cifra igual un mismo mensaje cada vez y a priori al adversario le es más complicado superar los retos CPA y CCA por comparación.

Esto se debe a que el atacante sabe qué par de funciones están siendo utilizadas por el algoritmo, y por tanto no hay duda de qué mensaje es el que ha sido cifrado. En este apartado presentamos una variante no determinista del esquema, y así evitar que un mensaje se cifre siempre del mismo modo.

Sencilla variante no determinista.

Vamos a presentar una sencilla variante no determinista del esquema, introduciendo aleatoriedad a la hora de ejecutar el algoritmo de cifrado, y vamos a estudiar si este cambio es significativamente mejor que la propuesta por los autores en [6].

De esta manera hacemos que en cada intercambio de mensajes se elija de forma aleatoria el par de funciones de cifrado y descifrado. Así para un mismo mensaje m, dependiendo del cuasigrupo en el que se esté empleando, hay hasta 3 posibles textos cifrados diferentes, y para cada uno de ellos hay hasta 2 formas de descifrarlos posibles.

De esta manera ya no se conoce cuál es la función que se emplea en el cifrado del mensaje, y además el cifrado de un mismo mensaje puede generar distintos textos cifrados, por lo que ahora no podríamos ganar el reto por comparación directa.

Este sencillo cambio, conlleva modificar tanto el algoritmo de generación de claves $\mathcal{G}$, como el de cifrado $\mathcal{E}$, ya que implica que la selección del par de funciones a utilizar no se lleve a cabo en $\mathcal{G}$, que ahora sólo genera la clave, sino en $\mathcal{E}$, ya que es a la hora de cifrar el texto claro m cuando se escogen dichas funciones.

Observación. Debido a que el par de funciones de cifrado y descifrado es diferente para cada mensaje que se envía, también habría que buscar una forma segura de intercambiar esta información de forma segura. Pero este es un punto en el que no vamos a profundizar. Supondremos a partir de ahora que estas condiciones se dan de forma adecuada.

7.2. Análisis de seguridad CPA y CCA en la variante no determinista del esquema

La pequeña mejora que introducimos en contraposición al determinismo no es una ventaja muy grande, ya que el esquema sigue siendo inseguro. Para empezar, hay cuasigrupos, como

el presentado anteriormente del Sistema de Steiner que, al sólo tener un par de funciones posible (S, S) debido a que $S = D = \hat{D}$, el esquema sigue siendo determinista y por tanto inseguro.

En el resto de cuasigrupos, si bien no se puede hacer una comparación directa, se puede seguir ganando por comparación a través de la ejecución de los oráculos un número polinómico de veces. Veámoslo con distintos ejemplos.

7.2.1. Ejemplos

Ejemplo 7.2.2 Sobre un espacio vectorial.

Utilizando el cuasigrupo basado en el Espacio Vectorial Finito del apartado 6.5, y sean la clave $k = (0, 1, 2)(3, 4, 4)$ y los dos mensajes $m_0 = (1, 1, 1)(2, 4, 3)$ y $m_1 = (2, 0, 3)(3, 0, 1)$. Ahora el oráculo $\mathcal{E}$ cifra utilizando aleatoriamente $S(m, k)$, $D(m, k)$ o $D(k, m)$. Los posibles cifrados para cada mensaje se recogen en la siguiente tabla.

	$S(m,k)$	$D(m,k)$	$D(k,m)$
m_0	$(3,1,4)(0,4,1)$	$(2,1,0)(1,4,2)$	$(4,1,3)(4,4,0)$
m_1	$(1,3,0)(3,2,0)$	$(4,4,4)(3,1,3)$	$(3,2,1)(3,3,2)$

Cada función tiene una probabilidad de $\frac{1}{3}$ de haber sido usada. Para poder ganar, simplemente tendríamos que usar repetidamente el oráculo de cifrado con uno de los mensajes.

Concretamente podemos calcular el numero de veces que hay que repetir el uso del oráculo para saber con exactitud si se trata de m_0 o m_1 a través de la siguiente fórmula.

Proposición 7.2.1 Número mínimo de veces que hay que usar el oráculo. *Sea $n \in \mathbb{N}$ el número de posibles funciones de cifrado distintas, y sea $m \in \mathbb{N}$ el número de veces que se ejecuta el oráculo $\mathcal{O}$. Sea $\mathcal{P}$ la probabilidad de que en las m repeticiones se haya cifrado un mensaje con las n funciones. Entonces, esta probabilidad viene dada por la ecuación:*

$$\mathcal{P} = 1 - \left[n \cdot \left(\frac{n-1}{n} \right)^m - n \cdot \left(\frac{1}{n} \right)^m \right]$$

Demostración:
Suponiendo que tenemos n cifrados distintos posibles $c_1, \ldots, c_n$ y que utilizamos el oráculo m veces. Queremos calcular la probabilidad de que se haya cifrado con los n cifrados.

Primero definimos los distintos sucesos:

$$\begin{aligned} s_0 &= \text{Se ha cifrado con los n cifrados distintos} \\ s_1 &= \text{Sólo se ha cifrado con } c_1 \\ \vdots &\ \vdots \\ s_n &= \text{Sólo se ha cifrado con } c_n \\ s_{n1} &= \text{No se ha cifrado con } c_1 \\ \vdots &\ \vdots \\ s_{nn} &= \text{No se ha cifrado con } c_n \end{aligned}$$

Entonces:

$$\begin{aligned} &i) \quad \Pr[s_0] = 1 - \Pr[\textbf{No } s_0] \\ &ii) \quad \Pr[\textbf{No } s_0] = \Pr[s_{n1}] + \ldots \quad + \Pr[s_{nn}] - (\Pr[s_1] + \cdots + \Pr[s_n]) \end{aligned}$$

Suponiendo que la elección del cifrado es totalmente aleatoria, tenemos la mismas probabilidades de que salga cada cifrado, y por tanto:

$$\begin{aligned} &a) \quad \Pr[s_1] = \cdots = \Pr[s_n] \\ &b) \quad \Pr[s_{n1}] = \cdots = \Pr[s_{nn}]. \end{aligned}$$

Usando combinatoria, sabemos que la probabilidad cifrar con sólo un cifrado c_i repitiendo m veces es $\Pr[s_i] = \frac{1}{n} \cdot \overset{\text{m veces}}{\cdots} \cdot \frac{1}{n} = (\frac{1}{n})^m$. Mientras que la probabilidad de que se usen todos los cifrados menos c_i es $\Pr(s_{ni}) = (\frac{n-1}{n})^m$.

Entonces nos queda que:

$$\mathcal{P} = \Pr[s_0] = 1 - [n \cdot \Pr[s_{ni}] - n \cdot \Pr(s_i)] =$$

$$= 1 - \left[n \cdot \left(\frac{n-1}{n} \right)^m - n \cdot \left(\frac{1}{n} \right)^m \right] \quad \square$$

Por tanto, dado un c' como reto, en este caso tenemos que con una probabilidad $\mathcal{P} = 0{,}99$, y con $n = 3$ funciones, basta usar el oráculo $\mathcal{E}$ únicamente 15 veces con m_0. Si las denotamos $\{c_1, \ldots, c_{15}\}$ podemos decir que si existe $i \in \{1, \ldots, 15\}$ t.q $c_i = c'$, entonces $b' = 0$, y en caso contrario, $b' = 1$.

Ejemplo 7.2.3 Si usamos ahora el grupo de permutaciones del apartado 6.2 con la clave $k = fg(id)j$ y los mensajes $m_0 = jihj$ y $m_1 = (id)hgf$ se utilizarán aleatoriamente cualquiera de los 6 pares de funciones disponibles. Podemos ver un resumen de cómo cifraría en cada caso cada mensaje en la siguiente tabla:

	$S(m,k)$	$S(k,m)$	$D(k,m)$	$\hat{D}(k,m)$	$D(m,k)$	$\hat{D}(m,k)$
m_0	$hhh(id)$	$ijh(id)$	$ijh(id)$	$hhh(id)$	$ijh(id)$	$hhh(id)$
m_1	$fjgi$	$figh$	$fifi$	$fjfh$	$gigi$	$gjgh$

En este caso, usando la fórmula anterior con $n = 6$, dado el c' del reto, con una probabilidad de 0,99 podríamos acertar qué mensaje es el que ha sido cifrado utilizando el oráculo 36 veces en el peor caso.

7.2.4. Colisiones

Hemos visto que la variante no determinista presentada es igual de insegura que la determinista, y el único reto que presenta es usar 35 veces más el oráculo, y este avance no supone ningún incremento en la seguridad del esquema. Aún con esto, puede haber ciertos momentos en el que el adversario no pueda distinguir de ninguna forma cuál de los mensajes ha sido cifrado. Esto ocurre cuando hay colisiones entre los mensajes, es decir, cuando dos mensajes diferentes tienen el mismo cifrado utilizando distintas funciones, y la misma probabilidad de que se cifren así. Veamos unos ejemplos de colisiones:

Ejemplo 7.2.5 En el cuadrado latino del apartado 6.3 tenemos una colisión con la siguiente configuración: $k = 3001$, $m_0 = 2023$, $m_1 = 0023$. Como hemos visto antes, solo tenemos disponibles los pares de cifrado (S, S), $(S, \hat{D})$ y $(\hat{D}, S)$, vamos a ver como queda la tabla de cifrados:

	$S(m,k)$	$S(k,m)$	$\hat{D}(m,k)$	$\hat{D}(k,m)$
m_0	0311	2311	2131	2131
m_1	2311	0311	0131	0131

Si el cifrado del reto es $c' = 0311$ o $c' = 2311$, el adversario no es capaz de distinguir si viene de m_0 o m_1.

Ejemplo 7.2.6 Volviendo al grupo de permutaciones presentado en 6.2 con la siguiente configuración: $k = fg(id)j$, $m_0 = ghff$ y $m_1 = (id)hgf$, tenemos los posibles cifrados:

	$S(m,k)$	$S(k,m)$	$D(k,m)$	$\hat{D}(k,m)$	$D(m,k)$	$\hat{D}(m,k)$
m_0	$(id)jfi$	$(id)ifh$	$gigi$	$gjgh$	$fifi$	$fjfh$
m_1	$fjgi$	$figh$	$fifi$	$fjfh$	$gigi$	$gjgh$

De donde podemos ver que hay 3 posibles colisiones: $c' = fifi$, $c' = gigi$ y $c' = gjgh$.

Es importante el hecho de que las probabilidades de que aparezca el mismo cifrado sean iguales, ya que en caso contrario se pueden llegar a obtener probabilidades de éxito mayores

del $0,5$, en cuyo caso suponemos que el adversario ha ganado por tener más información que si elige aleatoriamente.

Ejemplo 7.2.7 Por ejemplo, en el caso del grupo $(\mathbb{Z}_3, \oplus)$ de los ejemplos 5.3.6 y 6.1.1, si tenemos la configuración $k = 1210$, $m_0 = 0120$ y $m_1 = 2210$, nos da los siguientes posibles cifrados:

	$S(m,k)$	$D(k,m)$	$D(m,k)$
m_0	1000	1110	2210
m_1	0120	1000	1000

Supongamos que tenemos el reto $c' = 1000$. Sabemos, por tener 3 posibles cifrados, que ejecutando el oráculo $\mathcal{E}$ 15 veces, tenemos una probabilidad del $0,99$ de que hayan salido los 3 cifrados al menos una vez cada uno. Después de correr el algoritmo las 15 veces para m_0 y para m_1 se puede analizar en qué caso ha salido más veces $c = 1000$, y dado que $\mathbf{Pr}(\mathcal{E}(m_0) = 1000) = \frac{1}{3}$ y $\mathbf{Pr}(\mathcal{E}(m_1) = 1000) = \frac{2}{3}$, podemos afirmar con una seguridad de $\frac{2}{3} > \frac{1}{2}$ que el mensaje cifrado ha sido m_1.

A pesar de que existan ejemplos concretos en los que la certeza de acertar sea más baja de lo normal o inferior al $0,5$ de la decisión aleatoria, son casos minoritarios y no garantizan ningún tipo de seguridad. Es más, en ambos experimentos el adversario tiene acceso al oráculo antes de elegir los mensajes m_0 y m_1, por lo que puede ejecutarlo el número mínimo de veces para estar seguro de que han salido todos los posibles cifrados y de esta forma escoger dos mensajes que no presenten colisión.

7.3. Posible solución: Combinaciones lineales.

Una posible solución pasa por realizar combinaciones lineales entre todos los distintos cifrados disponibles para el cuasigrupo $(Q, \oplus)$.

Este cambio también implica toda una modificación en los algoritmos de generación de claves $\mathcal{G}$, así como en los de cifrado $\mathcal{E}$ y descifrado $\mathcal{D}$ sobre la anterior propuesta, presentada en la sección 7.1.

7.3.1. Abordaje general

Supongamos que, para un mensaje m, en lugar de elegir un par de funciones y cifrar con la correspondiente, $\mathcal{E}$ funciona del siguiente modo. Si n es el número de cifrados distintos que tenemos disponibles $f_1, \ldots, f_n$:

1. Se elige $(\lambda_1, \ldots, \lambda_n) \in Q^n$

2. Se cifra el mensaje con todos los cifrados: $c_i = f_i(m,k), \quad \forall i \in \{1,\dots,n\}$

3. Ahora hacemos la combinación lineal de todos los c_i, dando lugar al texto cifrado.

$$\mathcal{E}(k,m) = c = \sum_{i=1}^{n} \lambda_i \cdot c_i = \sum_{i=1}^{n} \lambda_i \cdot f_i(m,k).$$

4. Para recuperar el mensjae completo, tenemos que revertir la combinación lineal, por lo que serían necesarios los λ_i. Por tanto, el receptor debe recibirlos junto con el texto cifrado c. Hay distintas opciones: como pasarlos como parte del propio texto cifrado, o involucrando la clave privada k en su generación usando un generador pseudoaleatorio o emplear otro cifrado para intercambiarlos, de forma que sólo el destinatario pueda obtenerlos.

Observación 7.3.1 Nótese que se necesita tener un producto definido en Q para poder emplear este método de cifrado.

Veamos cómo puede implementarse esta idea a través de un ejemplo sobre $(\mathbb{Z}_3, \oplus, \cdot)$ suponiendo que queremos enviar $m = 0120$, con la clave $k = 1121$, que ha sido generada por $\mathcal{G}$.

Entonces tenemos que $\mathcal{E}$ genera primero $(\lambda_1, \lambda_2, \lambda_3) = (1,1,2)$, y después procede al cifrado $c_1 = S(m,k) = 1211$, $c_2 = D(k,m) = 1001$ y $c_3 = D(m,k) = 2002$. Por lo tanto el texto cifrado resulta $c = 1 \cdot (1121) \oplus 1 \cdot (1001) \oplus 2 \cdot (2002) = 0210$.

Ahora para descifrar $c = 0210$ conociendo únicamente $k = 1121$ y $(\lambda_1, \lambda_2, \lambda_3) = (1,1,2)$, tenemos que revertir la combinación lineal anterior:

$$c = 1 \cdot S(m,k) + 1 \cdot D(k,m) + 2 \cdot D(m,k).$$

Si despejamos, por ejemplo $S(m,k)$, tenemos $S(m,k) = c - D(k,m) - 2 \cdot D(m,k)$, y como $D(S(m,k),k) = m$ podemos ver que:

$$m = c - D(k,m) - 2 \cdot D(m,k) - k \iff m = c - (k-m) - 2 \cdot (m-k) = c - m \iff 2 \cdot m = c \iff$$

$$\iff m = \frac{c}{2} = \frac{0210}{2} = 0120.$$

Generalizando este proceso para cualquier terna $(\lambda_1, \lambda_2, \lambda_3)$ tenemos que:

$$c = \lambda_1 \cdot S(m,k) \oplus \lambda_2 \cdot D(k,m) \oplus \lambda_3 \cdot D(m,k) \iff S(m,k) = \frac{1}{\lambda_1} \cdot [c - \lambda_2 \cdot D(k,m) - \lambda_3 \cdot D(m,k)] \,.$$

Sustituyendo en $D(S(m,k),k) = m$:

$$m = \frac{1}{\lambda_1} \cdot [c - \lambda_2 \cdot D(k,m) - \lambda_3 \cdot D(m,k)] - k = \frac{1}{\lambda_1} [c - \lambda_2 \cdot k + \lambda_2 \cdot m - \lambda_3 \cdot m\lambda_3 \cdot k] - k =$$

$$= \frac{1}{\lambda_1} \cdot [c + k \cdot (\lambda_3 - \lambda_2) + m \cdot (\lambda_2 - \lambda_3)] - k = \frac{1}{\lambda_1} \cdot [c + k \cdot (\lambda_3 - \lambda_2)] + \frac{1}{\lambda_1} \cdot m \cdot (\lambda_2 - \lambda_3) - k \iff$$

$$\iff m - \frac{1}{\lambda_1} \cdot m \cdot (\lambda_2 - \lambda_3) = m \cdot [1 - \frac{1}{\lambda_1} \cdot (\lambda_2 - \lambda_3)] = \frac{1}{\lambda_1} \cdot [c + k \cdot (\lambda_3 - \lambda_2)] - k \iff$$

$$\iff m = \frac{1}{1 - \frac{1}{\lambda_1} \cdot (\lambda_2 - \lambda_3)} \cdot \left[\frac{1}{\lambda_1} \cdot (c + k \cdot (\lambda_3 - \lambda_2)) - k \right].$$

Para poder realizar este cálculo, es necesario que λ_1 tenga inverso, al igual que $1 - \frac{1}{\lambda_1} \cdot (\lambda_2 - \lambda_3)$. Por tanto, los cuasigrupos con los que podremos hacer estas combinaciones lineales para cifrar los textos han de ser cuerpos.

Adicionalmente, en la elección de la terna $(\lambda_1, \lambda_2, \lambda_3)$ hay que tener en cuenta que:

1. $\lambda_1 \neq 0$
2. $\lambda_2 - \lambda_3 \neq \lambda_1$.

Porque en caso contrario se produciría una división por 0.

Observación 7.3.2 Al trabajar sobre cuerpos, sólo tenemos 3 cifrados posibles y, por tanto, únicamente tenemos que generar 3 coeficientes λ_1, λ_2 y λ_3.

7.3.2. Ejemplo concreto.

En primer lugar hay que elegir un cuerpo finito donde podamos establecer todo este cifrado. En este ejemplo usaremos $F(5) = \mathbb{Z}_5$. Supongamos que queremos enviar el mensaje $m = 1134$.

El proceso completo de generación de claves, cifrado y descifrado es como sigue:

1. $\mathcal{G}$ genera la clave k y la terna $(\lambda_1, \lambda_2, \lambda_3)$ siguiendo las restricciones oportunas. En nuestro caso supondremos que $k = 2304$ y $(\lambda_1, \lambda_2, \lambda_3) = (4, 2, 1)$.
2. Primero $\mathcal{E}$ cifra m utilizando las 3 funciones diferentes:
   a) $S(m,k) = 1134 \oplus 2394 = 3433$
   b) $D(k,m) = 2394 \oplus 1134 = 1220$
   c) $D(m,k) = 1134 \oplus 2394 = 4330$

3. Después $\mathcal{E}$ realiza las combinaciones lineales entre los distintos cifrados:

$$c = \lambda_1 \cdot S(m,k) \oplus \lambda_2 \cdot D(k,m) \oplus \lambda_3 \cdot D(m,k) =$$

$$= 4 \cdot 3433 \oplus 2 \cdot 1220 \oplus 1 \cdot 4330 = 2122 \oplus 2440 \oplus 4330 = 3342.$$

4. De la misma forma, para realizar el descifrado, $\mathcal{D}$ aplica la formula:

$$m = \frac{1}{1 - \frac{1}{4} \cdot (2-1)} \cdot \left(\frac{1}{4} \cdot [3342 + 2304 \cdot (1-2)] - 2304 \right) =$$

$$= \frac{1}{2} \cdot (4012 - 2304) = \frac{1}{2} \cdot (2213) = 1134.$$

Capítulo 8

Conclusiones.

Este trabajo nos permite extraer diferentes conclusiones; algunas genéricas, que pueden aplicarse a la evaluación de otros esquemas criptográficos y otras vinculadas al esquema en estudio. De manera resumida, podemos enumerar las siguientes ideas fundamentales:

1. Un esquema de cifrado no es más seguro por emplear técnicas o estructuras algebraicas poco utilizadas. Los autores de la propuesta original apostaban por los cuasigrupos debido a que su uso en los sistemas criptográficos no ha sido muy frecuente, y también dada su limitada estructura y poca complejidad. Sin embargo, hemos visto que esto no quiere decir que un esquema basado en ellos sea resistente a ataques sencillos. Es más, en cuanto se ha reforzado un poco el sistema haciendo uso de las combinaciones lineales, hemos acabado utilizando una estructura con mayor complejidad como es un cuerpo.

2. Una herramienta potencialmente interesante, como son los cuasigrupos, per se, no aporta seguridad ni robustez. Hay que combinar estas estructuras con un buen diseño de los algoritmos. En el esquema propuesto, los algoritmos tienen grandes deficiencias o fallos, principalmente por ser deterministas.

3. Cuando perseguimos seguridad semántica, la realeatorización de un esquema de cifrado determinista, aunque mejora las propiedades originales del sistema, no aporta un avance suficiente en la protección del mismo. Pues con un sencillo ataque estadístico es posible determinar qué cifrado ha sido escogido con altas probabilidades.

4. Para trabajos a futuro, se puede seguir explorando la idea de construir un esquema de cifrado basada en cuasigrupos, ya que, como afirman los autores del esquema estudiado, es un campo en el que se ha indagado poco. Aunque, como ya se ha mencionado, la herramienta o estructura en la que se base dicho esquema debe ir acompañado de un buen diseño.

Capítulo 9

Anexo

9.1. Código de las implementaciones.

9.1.1. Algoritmo voraz de probabilidad.

A modo de facilitar los cálculos para saber el mínimo número de veces que hay que ejecutar el oráculo para romper el cifrado con una certeza de probabilidad dada y así calibrar el coste de la seguridad CCA y CPA, se facilita un algoritmo voraz escrito en Python.

El algoritmo está basado en la fórmula presentada en la proposición 7.2.1.

Código en Python: Cálculo de la probabilidad de repetir cifrado.

```
def probability_function(n, m):
    """
    Calculates the probability, using the provided formula:
    P = 1 - [n * ((n-1)/n)^m - n * (1/n)^m]
    Where:
        - n: is the number of possible cypher functions.
        - m: number of times the Oracle has been executed.
        - P: probability of getting all the possible cyphers under
             the n functions.
    """
    return 1 - (n * pow((n-1)/n, m) - n * pow(1/n, m))


def min_elements(n, required_probability):
    """
    Simple greedy algorithm that finds the minimum number of tries an
    attacker would need to get all the possible cypher-texts under all
    :n: possible functions, with a probability of :required_probability:
    """
```

```
    current_probability = 0
    i = 0
    while current_probability <= required_probability:
        i += 1
        current_probability = probability_function(n, i)
    return i


elements_number = 10
probability = 0.99

print(f"Minimum number of tries with {elements_number} functions to get"
      f" a {probability} probability: {min_elements(10, 0.99)} tries.")
```

9.1.2. Programa para el manejo de Cuasigrupos.

Para ayudar a la comprobación y elaboración de las distintas tablas de los cuasigrupos, y facilitar las cuentas y los cálculos de cifrados y descifrados bajo distintas condiciones, se ha desarrollado un pequeño programa Python.

Este programa ofrece funcionalidad para calcular todas las tablas requeridas durante el trabajo, así como opciones para aplicar las distintas funciones S, D, y $\hat{D}$. Para el cálculo de tablas se hace uso de la proposición 5.2.4.

También incluye las fórmulas presentadas para el cifrado y descifrado usando combinaciones lineales sobre cuerpos, tal y como se especifica en el apartado 7.3.

La especificación del cuasigrupo o del cuerpo se introduce a través de un fichero de texto con el formato correcto, que es el siguiente:

- La primera linea está compuesta por 3 elementos separados por un espacio:
  - Tipo de Objeto: QG si es un cuasigrupo o F si es un cuerpo.
  - Nombre: Nombre del objeto (sin espacios).
  - Numero de elementos que tiene.
- La siguiente linea contiene los elementos del cuasigrupo o cuerpo.
- Las siguientes n lineas (donde n es el número de elementos) son la tabla de la operación del cuasigrupo, o la tabla de la suma del cuerpo.
- Si estamos especificando un cuerpo, además habrá n lineas adicionales, indicando la tabla de producto del cuerpo.

A continuación tenemos un ejemplo de 3 especificaciones, dos de cuasigrupos ($\mathbf{Z}_3$ y S_3)y una de un campo ($\mathbf{Z}_5$)

Especificación de Cuasigrupos y Cuerpos en el fichero de texto.

```
QG S3 6
(id), f, g, h, i, j
(id), f, g, h, i, j
f, g, (id), j, h, i
g, (id), f, i, j, h
h, i, j, (id), f, g
i, j, h, g, (id), f
j, h, i, f, g, (id)

QG Z3 3
0, 1, 2
0, 1, 2
1, 2, 0
2, 0, 1

F Z5 5
0, 1, 2, 3, 4
0, 1, 2, 3, 4
1, 2, 3, 4, 0
2, 3, 4, 0, 1
3, 4, 0, 1, 2
4, 0, 1, 2, 3
0, 0, 0, 0, 0
0, 1, 2, 3, 4
0, 2, 4, 1, 3
0, 3, 1, 4, 2
0, 4, 3, 2, 1
```

Código en Python: Funcionalidad para trabajar con Cuasigrupos.

```
import random

"""
This is a demo implementation of Quasigroups, Fields and main
functionality of Linear Combinations with the presented Cyphers.
"""


class QuasiGroup:
    def __init__(self, name, elements, operation_table):
        self.name = name
        self.elements = elements
        self._operation_table = operation_table

        self._s_function = Utils.calc_s_function(operation_table,
                                                 elements)
        self._d_function = Utils.calc_d_function(self._s_function,
```

```
18                                                  elements)
19         self._d_hat_function = Utils.calc_d_hat_function(
20             self._s_function, elements)
21
22     def _get_name(self):
23         return f"Quasigroup {self.name}\n"
24
25     def __str__(self):
26         output = self._get_name()
27         output += self.print_s_table()
28         output += self.print_d_table()
29         output += self.print_dh_table()
30         return output
31
32     def s(self, a, b):
33         return self._s_function[(a, b)]
34
35     def d(self, a, b):
36         return self._d_function[(a, b)]
37
38     def dh(self, a, b):
39         return self._d_hat_function[(a, b)]
40
41     # Cyphering methods that apply the function to lists
42     # of elements of the QG
43     @staticmethod
44     def _apply(f, a, b):
45         result = list()
46
47         for i in range(len(a)):
48             result.append(f(a[i], b[i]))
49         return result
50
51     def apply_s(self, a, b):
52         return self._apply(self.s, a, b)
53
54     def apply_d(self, a, b):
55         return self._apply(self.d, a, b)
56
57     def apply_dh(self, a, b):
58         return self._apply(self.dh, a, b)
59
60     # Printing methods that show the tables.
61     def _print_table(self, table, table_name, div=False):
62         """
63         div=True skips printing division by 0.
64         """
```

```
65          output = f"{table_name}\n"
66
67          for element_i in self.elements:
68              for element_j in self.elements:
69                  if div and element_j == 0:
70                      output += "X\t"
71                      continue
72
73                  output += f"{table[(element_i, element_j)]}"
74                  if element_j != self.elements[-1]:
75                      output += "\t"
76              output += "\n"
77          output += "\n"
78          return output
79
80      def print_s_table(self):
81          return self._print_table(self._s_function, "S")
82
83      def print_d_table(self):
84          return self._print_table(self._d_function, "D")
85
86      def print_dh_table(self):
87          return self._print_table(self._d_hat_function, "^D")
88
89
90  class Field(QuasiGroup):
91      def __init__(self, name, elements, sum_table, product_table):
92          super(Field, self).__init__(name, elements, sum_table)
93          self._product_table = product_table
94          self._product_function = Utils.calc_s_function(product_table,
95                                                         elements)
96          self._division_function = Utils.calc_d_function(
97              self._product_function, elements)
98
99      def _get_name(self):
100         return f"Field {self.name}\n"
101
102     def __str__(self):
103         output = super(Field, self).__str__()
104         output += self._print_table(self._product_function, "x")
105         output += self._print_table(self._division_function, "/",
106                                     div=True)
107         return output
108
109     @staticmethod
110     def _apply_scalar_vector_function(f, scalar, vector):
111         """
```

```
112         In fields, apply_p and apply_div are functions that receive an
113         scalar and a vector and multiplies (resp. divides) the vector
114         by the scalar.
115 
116         This is because element by element product or division is not
117         used.
118         """
119         result = list()
120         for element in vector:
121             result.append(f(scalar, element))
122         return result
123 
124     def product(self, a, b):
125         return self._product_function[(a, b)]
126 
127     def division(self, a, b):
128         return self._division_function[(a, b)]
129 
130     def apply_p(self, a, b):
131         return self._apply_scalar_vector_function(self.product, a, b)
132 
133     def apply_div(self, a, b):
134         return self._apply_scalar_vector_function(self.product, a, b)
135 
136 
137 class Utils:
138     @staticmethod
139     def calc_s_function(operation_table, elements):
140         """
141         We refactor the operation table into a map function.
142         """
143         elements_number = len(elements)
144         function = dict()
145 
146         for i in range(elements_number):
147             for j in range(elements_number):
148                 function[(elements[i], elements[j])] = \
149                 operation_table[i][j]
150         return function
151 
152     @staticmethod
153     def calc_d_function(s_function, elements):
154         """
155         We calc D(x,y) with:    D(x,y) = z <=> S(z,y) = x
156         """
157         function = dict()
158         for element_i in elements:
```

```
159             for element_j in elements:
160                 function[(s_function[(element_i, element_j)],
161                          element_j)] = element_i
162         return function
163 
164     @staticmethod
165     def calc_d_hat_function(s_function, elements):
166         """
167         We calc ^D(x,y) with:     ^D(x,y) = z <=> S(y,z) = x
168         """
169         function = dict()
170         for element_i in elements:
171             for element_j in elements:
172                 function[(s_function[(element_j, element_i)],
173                          element_j)] = element_i
174         return function
175 
176     @staticmethod
177     def load_data(text_file, integer_mode=False):
178         """
179         Loads the cuasigroup from a text_file with the correct format.
180         Integer_mode is for the case that the Quasigroup is made of
181         integers.
182         """
183 
184         def process(x):
185             return int(x.strip()) if integer_mode else x.strip()
186 
187         def read_table():
188             table = list()
189             for i in range(elements_number):
190                 table_line = list(map(lambda x: process(x),
191                                       f.readline().split(",")))
192                 table.append(table_line)
193             return table
194 
195         with open(text_file) as f:
196             line = f.readline().split()
197             is_field = (line[0] == "F")
198             name = line[1]
199             elements_number = int(line[2])
200             elements = list(
201                 map(lambda x: process(x), f.readline().split(",")))
202             operation_table = read_table()
203             if is_field:
204                 product_table = read_table()
205                 return Field(name, elements, operation_table,
```

```
206                                 product_table)
207
208         return QuasiGroup(name, elements, operation_table)
209
210     @staticmethod
211     def parse_construction(qg, construction, integer_mode=False):
212         """
213         This method receives a String as input, and it outputs a list
214         with each element of the Quasigroup in that string.
215         """
216         elements_set = set(qg.elements)
217         output = list()
218         temp = ""
219
220         for char in construction:
221             temp += char
222             if integer_mode:
223                 temp = int(temp)
224             if temp in elements_set:
225                 if integer_mode:
226                     output.append(int(temp))
227                 else:
228                     output.append(temp)
229                 temp = ""
230         return output
231
232
233 class LinearCombination:
234     """
235     This class is used to show the functionality of the Linear
236     Combination Cypher described.
237     """
238
239     def __init__(self, field):
240         self._field = field
241
242     def create_random_seed(self):
243         random_seed = [random.choices(self._field.elements) for _ in
244                        range(3)]
245         return random_seed
246
247     def encrypt(self, message, key, random_seed):
248         # There are these 3 different ways of cypher.
249         s_cypher = self._field.apply_p(random_seed[0],
250                                        self._field.apply_s(message,
251                                                            key))
252         flipped_d_cypher = self._field.apply_p(random_seed[1],
```

```
253                                                 self._field.apply_d(
254                                                     key, message))
255         d_cypher = self._field.apply_p(random_seed[2],
256                                        self._field.apply_d(message,
257                                                            key))
258
259         return self._field.apply_s(
260             self._field.apply_s(s_cypher, flipped_d_cypher), d_cypher)
261
262     def decrypt(self, cypher_text, key, random_seed):
263         constant_1 = self._field.division(1, random_seed[0])
264         constant_2 = self._field.d(random_seed[1], random_seed[2])
265         constant_3 = self._field.d(0, constant_2)
266
267         left_member = self._field.product(constant_1, constant_2)
268         left_member = self._field.d(1, left_member)
269         left_member = self._field.division(1, left_member)
270
271         right_member = self._field.apply_p(constant_3, key)
272         right_member = self._field.apply_s(cypher_text, right_member)
273         right_member = self._field.apply_p(constant_1, right_member)
274         right_member = self._field.apply_d(right_member, key)
275
276         return self._field.apply_p(left_member, right_member)
277
278
279 def basic_example_test():
280     qg = Utils.load_data("./quasi_group_descriptions.txt")
281     print(qg)
282     # The result of adding 2 + 2 in the QG
283     print(qg.s("2", "2"))
284     # Cyphering 0012 + 1111
285     print(qg.apply_s(["0", "0", "1", "2"], ["1", "1", "1", "1"]))
286     # The same as:
287     print(qg.apply_s(Utils.parse_construction(qg, "0012"),
288                      Utils.parse_construction(qg, "1111")))
289
290
291 def qg_example():
292     qg = Utils.load_data("./quasi_group_descriptions.txt")
293     print(qg)
294     m = Utils.parse_construction(qg, "hihi")
295     k = Utils.parse_construction(qg, "fg(id)j")
296
297     cypher_text = qg.apply_s(m, k)
298     print(f"Cypher text: {cypher_text}")
299     m_p = qg.apply_d(cypher_text, k)
```

```
300     print(f"Recovered Plain Text: {m_p}")
301
302
303 def linear_combinations_test():
304     field = Utils.load_data("./quasi_group_descriptions.txt",
305                             integer_mode=True)
306     print(field)
307     lc = LinearCombination(field)
308     random_seed = Utils.parse_construction(field, "421",
309                                            integer_mode=True)
310     message = Utils.parse_construction(field, "1134",
311                                        integer_mode=True)
312     key = Utils.parse_construction(field, "2304", integer_mode=True)
313
314     c = lc.encrypt(message, key, random_seed)
315     print(
316         f"With the message {message}, the key {key} and the random "
317         f"seed {random_seed} we have the Cypher text: {c}")
318     m_prime = lc.decrypt(c, key, random_seed)
319     print(f"Decrypted {m_prime} should be equal to message {message}")
320
321
322 basic_example_test()
323 linear_combinations_test()
324 qg_example()
```

Bibliografía

[1] Quasi-group. encyclopedia of mathematics., s.f Recuperado en Julio de 2019.

[2] Shafi Goldwasser and Mihir Bellare. Lecture notes on cryptography. *Summer course "Cryptography and computer security" at MIT*, 2008.

[3] Thomas W. Hungerford. *Algebra*. Springer, 1974.

[4] Jonathan Katz. *Introduction to modern cryptography / Jonathan Katz, Yehuda Lindell*. Chapman & Hall/CRC cryptography and network security. Chapman & Hall/CRC, Boca Raton [etc.], 2nd ed. edition, 2015.

[5] C Koscielny. A method of constructing quasigroup-based stream-ciphers. *Applied Mathematics and Computer Science*, 6:109–122, 1996.

[6] Czeseaw Kościelny and GL Mullen. A quasigroup-based public-key cryptosystem. *International Journal of Applied Mathematics and Computer Science*, 9(4):955–963, 1999.

[7] Gregory T. Lee. *Abstract Algebra*. Springer, 2018.

[8] Nigel Paul Smart et al. *Cryptography: an introduction*, volume 3. McGraw-Hill New York, 2003.

[9] Jonathan-D Smith. Four lectures on quasigroup representations. *Quasigroups and Related Systems*, 15(1):109–140, 2007.

[10] Jonathan DH Smith. *An introduction to quasigroups and their representations*. CRC Press, 2006.