FACULTAD DE INFORMÁTICA DE BARCELONA (FIB)

UNIVERSIDAD POLITÉCNICA DE CATALUNYA (UPC)

GRADO EN INGENIERÍA INFORMÁTICA

TECNOLOGÍAS DE LA INFORMACIÓN

Criptografía post-cuántica y códigos correctores de errores

Memoria del proyecto

Autor: Daniel Medina Tolrá

Fecha de la defensa: 25/04/2019

supervisado por

Director: José Luis Ruiz (Departament de Matemàtiques)

Resumen

Este proyecto se encarga principalmente de estudiar los códigos binarios de Goppa y el criptosistema de McEliece (1978) como propuesta de un sistema criptográfico post-cuántico. Se estudia en profundidad su funcionamiento desde un punto de vista teórico y usando ejemplos simples que ayudan a su comprensión. También se estudia su seguridad de modo teórico y práctico, analizando y mencionando métodos de ataques ya conocidos. Al final del proyecto se pueden encontrar partes de la implementación del algoritmo corrector de errores usado y del criptosistema de McEliece.

Además, hay una propuesta basada en este criptosistema en un concurso del NIST[1]. Dicho concurso pretende encontrar un algoritmo lo suficientemente seguro para soportar ataques de ordenadores cuánticos y usarlo de estándar en la seguridad informática (como actualmente ocurre con el método RSA[2]). Es por este motivo que analizamos esta propuesta y explicamos los resultados obtenidos en su estudio donde podremos ver, entre otras cosas, la eficiencia de los algoritmos. Por el momento esta propuesta ya ha pasado varias rondas de control y parece ser bastante prometedora. Pero como todo criptosistema, tiene sus ventajas y limitaciones. Así que aún será necesario poner todas las propuestas a prueba durante un tiempo para poder escoger la definitiva.

Resum

Aquest projecte s'encarrega principalment d'estudiar els codis binaris de Goppa i el criptosistema de McEliece(1978) com a proposta d'un sistema criptogràfic post-quàntic. Se estudia en profunditat el funcionament des d'un punt de vista teòric y utilitzant exemples simples que ajuden a la comprensió. També s'estudia la seva seguretat de manera teòrica i pràctica, analitzant i mencionant formes d'atacar al sistema ja conegudes. Al final del projecte es poden trobar parts de la implementació del algoritme corrector d'errors utilitzat i del criptosistema de McEliece.

A més, hi ha una proposta basada en aquest criptosistema en un concurs del NIST. Aquest concurs té l'objectiu de trobar un algorisme que sigui suficientment segur per soportar atacs d'ordinadors quàntics i utilitzar-ho com a estàndard en la seguretat informàtica (com actualment passa amb el mètode RSA). Per aquest motiu s'analitza aquesta proposta i s'explican els resultats obtinguts en l'estudi on es pot veure, entre

[1] NIST (National Institute of Standards and Technology) (*1.2.1*)
[2] RSA (*1.1*)

altres coses, l'eficiència dels algorismes. De moment, la proposta ja ha passat varies rondes de control i sembla ser bastant prometedora. Pero com tot criptosistema, té avantatges i limitacions. Així que encara serà necessari posar totes les propostes a prova durant un temps per poder triar la definitiva.

Abstract

The main objective of this project is to study the binary Goppa code and the McEliece cryptosystem (1978) as a proposal of a post quantum cryptosystem. It is deeply analyzed and studied in a theoretical way and with some examples that contribute to its understanding. Its security is also studied in theory and practice analyzing some attack methods already known. At the end of the project can be found certain parts of the implementation of the error-correcting algorithm and the McEliece cryptosystem.

Furthermore, there is a proposal based on this cryptosystem in a Post-Quantum Cryptography Standardization Process of the NIST[3]. This standardization process aims to find an algorithm which is secure enough to stand attacks by quantum computers. For that reason, the methods and results obtained of the proposal are analyzed on this project. For the time being, this proposal has already passed some rounds and it seems to be promising. However, as with any cryptosystem, it has its advantages and limitations. It will take some time to test all proposals to be able to choose the definitive one.

[3] NIST (National Institute of Standards and Technology) (*1.2.1*)

ÍNDICE

1. INTRODUCCIÓN

1.1 Formulación del problema

La seguridad informática es el área encargada de la protección de las infraestructuras informáticas y, especialmente, de la privacidad y la integridad de la información almacenada. Como es sabido, el uso de las tecnologías de la información y comunicaciones es cada vez mayor y el número de ciberataques crece exponencialmente a diario; y poder llegar a revelar, corromper o incluso modificar cierta información de terceros puede tener graves consecuencias. Por lo tanto, la seguridad informática se convierte en uno de los principales problemas de hoy en dia de las empresas, organizaciones y gobiernos, entre otros; y tener una buena seguridad es imprescindible en cualquier sistema informático.

La criptografía es la ciencia encargada de estudiar las distintas técnicas usadas para transformar (cifrar) la información con el objetivo de hacerla irreconocible a receptores no autorizados. Por lo tanto, es una parte muy importante de la seguridad informática. El objetivo de la criptografía es proteger la información encriptándola, y que solo el destinatario de la información sea capaz de recuperar (desencriptar) la información original. Pero como todos sabemos, no hay nada seguro al 100%; y hay algoritmos y mecanismos que pueden intentar desencriptar la información sin conocer la clave para hacerlo. Entonces, es muy importante usar algoritmos de cifrado que sean muy difíciles de "romper".

A pesar de que nada es seguro del todo, hoy en día se usan sistemas criptográficos muy seguros. Uno de los más seguros y más usados es el sistema RSA. Este sistema fué desarrollado en 1977 por Rivest, Shamir y Adleman y su seguridad radica en el problema de la factorización de números enteros. Su funcionamiento se basa en hacer el producto de dos números primos elegidos al azar y mantenidos en secreto. Encontrar estos dos números (del orden de 10 elevado a 200) a partir del producto es algo demasiado costoso para los ordenadores de hoy en día. El sistema RSA es un sistema muy seguro a largo plazo ya que el tamaño de los dos primos puede hacerse mayor a medida que aumente la capacidad de cálculo de los ordenadores.

Se ha demostrado (de forma teórica) que la computación cuántica resolverá el problema de la factorización (algoritmo de Shor [4]) por lo que el RSA dejaría de ser seguro. Este nuevo tipo de computación está siendo desarrollada actualmente y es completamente distinta a la computación clásica actual. En la computación cuántica se usan qubits en

lugar de bits, y puertas lógicas distintas que hacen posibles nuevos algoritmos. Uno de ellos es el algoritmo de Shor ya mencionado, capaz de factorizar un número entero, y por tanto romper el sistema RSA, en tiempo polinómico. Esto supone un grave problema, ya que una vez hayan ordenadores cuánticos operativos (en un futuro próximo), necesitaremos un nuevo sistema criptográfico que sea seguro frente a la computación cuántica.

1.2 Contexto

1.2.1 NIST

El Instituto Nacional de Normas y Tecnología (llamado NIST por sus siglas en inglés, National Institute of Standards and Technology) es una agencia federal no regulatoria del Departamento de Comercio de los Estados Unidos. El objetivo principal de este instituto es promover la innovación y la competencia industrial. El NIST abarca muchas áreas y actividades diferentes como la nanotecnología, las tecnologías de la información, la biotecnología, la investigación con neutrones, entre otras.

El NIST, para solucionar el problema mencionado anteriormente, ha organizado un concurso público para buscar una solución. Dicho concurso, llamado "Post-Quantum Cryptography Standardization Process" [23], es muy importante para la seguridad informática en un futuro próximo, ya que pretende encontrar uno o más nuevos sistemas criptográficos de clave pública capaces de ser seguros a ataques con ordenadores cuánticos. La fecha límite para presentar propuestas fue el 30 de noviembre de 2017. Se han presentado decenas de propuestas diferentes a lo largo de 2017 y se han seleccionado las mejores. Se prevé que estas propuestas se pondrán a prueba durante los próximos años y se irán descartando las que no sean seguras, o mejorándolas si fuera posible.

1.2.2 Términos, conceptos y objetivo del trabajo

1.2.2.1 Propuesta elegida

El objetivo principal del trabajo es seleccionar una de las propuestas presentadas al concurso del NIST y estudiarla a fondo. La idea escogida se llama "*Classic McEliece: conservative code-based cryptography*" [13]. Se ha escogido esta propuesta por varios motivos:

- Parece ser bastante prometedora. Aún no ha sido descartada ni se ha conseguido vulnerar su seguridad.
- Es una propuesta basada en códigos correctores de errores. Es este un tema muy estudiado en la asignatura de "Transmisión y Codificación de la Información (TCI)".
- No es de las propuestas más complejas, y parece asequible llegar a estudiarla en profundidad (o incluso llegar a hacer una implementación sencilla) en el tiempo previsto para el trabajo.
- En general, se ha escogido este tema porque es un tema que puede llegar a ser muy relevante en el mundo de la seguridad informática en un futuro próximo.

Autores de la propuesta:

- Daniel J. Bernstein, University of Illinois at Chicago
- Tung Chou, Osaka University
- Tanja Lange, Technische Universiteit Eindhoven
- Ingo von Maurich, self
- Rafael Misoczki, Intel Corporation
- Ruben Niederhagen, Fraunhofer SIT
- Edoardo Persichetti, Florida Atlantic University
- Christiane Peters, self
- Peter Schwabe, Radboud University
- Nicolas Sendrier, Inria
- Jakub Szefer, Yale University
- Wen Wang, Yale University

1.2.2.2 Introducción al criptosistema de McEliece

La propuesta está basada en el algoritmo de McEliece de 1978, que es un algoritmo de corrección de errores. Puede parecer que este tipo de algoritmos no tengan nada que ver con la criptografía; pero si se añaden bastantes errores a un mensaje, éste sólo podrá ser leído si se conoce la manera de corregir los errores (si conoces la "clave"). Por lo tanto, pueden ser algoritmos muy útiles en la criptografía. Es por este motivo, que hay bastantes propuestas basadas en códigos.

Es un criptosistema que pretende ocultar el mensaje añadiendo errores; y el único modo de poder corregirlos es con una matriz generadora G (la clave privada). Además introduce una matriz binaria invertible S y una matriz de permutaciones P; y la clave

pública es el producto de las tres matrices SGP (también debe ser público el número máximo de errores corregibles).
A pesar de ser un sistema bastante eficiente y rápido, su clave tiene un tamaño demasiado grande, por lo que apenas recibe uso actualmente.. Además, tiene un factor de expansión del mensaje bastante elevado. Por lo tanto, no es demasiado útil en la computación clásica; pero parece tener futuro en la computación cuántica ya que el algoritmo de Shor no afecta a este sistema.

1.2.2.3 Temas principales del trabajo

No es un tema sencillo, y para llegar a estudiar la propuesta en profundidad, será necesario estudiar varios temas antes de comenzar con la propuesta. Estos son los temas principales del trabajo:

1. Códigos de Goppa binarios: Se tendrá que estudiar en profundidad cómo funcionan los códigos de Goppa, ya que son muy eficientes en la corrección de errores y son usados en el algoritmo de McEliece.
2. Criptosistema de McEliece: Se tendrá que conocer al detalle cómo funciona este criptosistema para poder llegar a entender la propuesta. Ya que los autores de la propuesta "rescataron" este algoritmo de 1978 y basaron en él su trabajo.
3. Computación cuántica: Antes de comenzar a estudiar la propuesta, se deberá entender como funciona la computación cuántica. Estamos acostumbrados a trabajar y estudiar con la computación clásica; por lo tanto, puede llegar a ser un tema complejo. Además es un tema muy extenso, pero creemos que tampoco será necesario llegar a estudiar la computación cuántica con mucha profundidad para entender la propuesta.
4. Propuesta: Por último, estudiar a fondo la propuesta. Además, intentar hacer una implementación (probablemente en Sage, ya que facilita mucho el código porque tiene muchas funciones relacionadas con los cuerpos finitos). Este apartado puede llegar a extenderse tanto como se pueda; se puede intentar encontrar algún punto débil, alguna pequeña mejora del sistema, razonar si llega a ser un sistema eficiente, ponerlo a prueba con diferentes parámetros si la implementación llega a funcionar correctamente, etc.

1.2.3 Actores implicados

Este trabajo va dirigido principalmente a la comunidad de expertos en seguridad informática, criptografía e incluso en matemáticas aplicadas a algoritmos de transmisión y corrección de errores. También puede ser muy útil para estudiantes, ya que parte de una base bastante más baja que la propuesta en sí; antes de analizarla habla en profundidad sobre códigos correctores de errores (en concreto el criptosistema de McEliece), sobre códigos de Goppa y sobre la computación cuántica. No se necesita ser un experto en computación cuántica para llegar a entender el trabajo.

Puede llegar a tener muchos usos como:

- Aprender y analizar a fondo el criptosistema de McEliece, los códigos de Goppa y la computación cuántica.
- Analizar a fondo la propuesta elegida partiendo de una base bastante más baja que en la propuesta elegida.
- Estudiar la propuesta para poder encontrar puntos débiles (ya que aún está en períodos de prueba); y de esta manera poder reforzar esos puntos débiles.
- Llegar a entender la propuesta a fondo para intentar mejorarla en cuanto a seguridad o eficiencia (y de esta forma poner a prueba estas mejoras más adelante).
- Comprender todos los temas relacionados a la propuesta para poder llegar a proponer otra idea parecida o llegar a darle algún otro uso diferente a un criptosistema.
- Entender paso a paso una implementación del criptosistema de la propuesta.

1.3 Estado del arte

Como se ha mencionado anteriormente, la computación cuántica está aún en desarrollo y se cree que llegará a ser una realidad en varios años. Es una computación con muchísimo potencial y ya hay varios prototipos actualmente. Por ejemplo, a principios de marzo de 2018, Google anunció el desarrollo de un nuevo procesador cuántico experimental llamado Bristlecone. Es el más potente registrado hasta el momento, con una cantidad de 72 Qubits. Pero para Google eso no es lo importante; el principal problema de estos procesadores cuánticos es el porcentaje de errores en las operaciones de lectura y lógica. Por lo tanto, aún serán necesarios varios años de desarrollo y pruebas para finalmente conseguir un ordenador cuántico funcional.

Un proceso como el del concurso público lanzado por el NIST requiere cierto tiempo, probablemente años. Ya que se deben poner a prueba todas las propuestas presentadas, atacarlas una vez tras otra; y reforzarlas, si es posible, en caso de tener debilidades. Llegar a dar con la mejor opción (o varias opciones) de todas es un proceso complicado y muy lento. Es por este motivo, que se debe hacer con suficiente antelación. Además, hasta que no existan ordenadores cuánticos operativos, no se podrán hacer pruebas concluyentes definitivas. Pero a pesar de no poder hacer demasiadas pruebas prácticas, se conoce bastante bien el funcionamiento teórico de la computación cuántica como para ir poniendo a prueba las propuestas presentadas al concurso.

Tras un proceso de selección se han elegido 71 propuestas para poner a prueba en el concurso del NIST. En tan solo 3 meses ya se han atacado 12 propuestas que serán retiradas si no se refuerzan; ya se han retirado 5 de las 12. Hay propuestas de varios tipos diferentes:

- 19 propuestas basadas en códigos.
- 29 propuestas basadas en retículos (lattices).
- 2 propuestas basadas en “hash”.
- 12 propuestas basadas en criptografía multivariante cuadrática.
- 9 propuestas basadas en otros métodos.

En resumen, la computación cuántica está en pleno desarrollo y el proceso de selección de un nuevo criptosistema para la criptografía cuántica del NIST está aún en la primera ronda (lleva solo 3 meses). El trabajo está basado en una de las propuestas que está siendo probada y estudiada actualmente. Este estudio está en pleno estado del arte.

1.4 Alcance

El alcance del proyecto es bastante difícil de predecir, ya que es un tema muy extenso y puede llegar a complicarse bastante. El alcance previsto es llegar a estudiar y analizar los 4 temas especificados en el apartado *1.2.2.3*. El tema 4 (estudiar a fondo la propuesta) se pretende extender tanto como se pueda. Se cree bastante probable hacer una implementación del criptosistema; pero con parámetros bastante más pequeños y realistas para poner a prueba con una computación clásica. Como ya se ha mencionado, estos estudios son mucho más teóricos que prácticos y es muy difícil poner a prueba los sistemas.

Problemas que podrían darse en el proyecto:

- Dificultad en la parte de computación cuántica.
- Dificultad en la implementación del criptosistema.
- Problemas en los parámetros de la posible implementación del criptosistema.
- Que la propuesta sea más complicada de entender de lo previsto.

1.5 Metodología y rigor

1.5.1 Método de trabajo

La metodología usada para el desarrollo del proyecto será en forma de cascada. Esta metodología consiste en desarrollar todas las fases del proyecto de manera iterativa. El proyecto cuenta con 4 fases o apartados principales: Códigos de Goppa binarios, criptosistema de McEliece 1978, computación cuántica y estudio de la propuesta. El objetivo principal del proyecto consiste en el estudio de la propuesta; y para poder analizar y entender la propuesta a fondo, se deben desarrollar los otros 3 apartados préviamente. Es por este motivo que se debe usar una metodología secuencial. El estudio de los códigos de Goppa debe ir antes que el estudio del criptosistema de McEliece, ya que este criptosistema usa estos códigos. Por otro lado, el apartado 3 (computación cuántica) no requiere un estudio previo de los apartados 1 y 2; tiene una temática bastante distinta. Pero se considera que los apartados 1 y 2 son más prioritarios y deben realizarse antes para evitar riesgos. Por lo tanto, el apartado 3 se podría desarrollar en paralelo con el 1 y 2. Pero como solo hay un trabajador en el proyecto, es mejor desarrollar los temas de uno en uno para centrar su máxima atención.

También se debe mencionar que antes de comenzar a desarrollar los apartados, se emplearán las primeras semanas del proyecto para leer y ponerse al día sobre los tres primeros apartados, ya que son temas que no se han visto demasiado durante la carrera. Una vez desarrollados los tres primeros apartados, ya se podrá comenzar a desarrollar el apartado del estudio de la propuesta(4). Éste último apartado probablemente ocupará una gran parte del proyecto y tendrá un subapartado con bastante peso que consistirá en hacer una implementación del criptosistema.

1.5.2 Métodos de validación

Se realizarán reuniones semanales con el director del proyecto para que éste compruebe que todo está yendo como es debido. Esto será muy necesario ya que es un tema bastante complejo; y si no se entiende a la perfección alguno de los apartados, será muy difícil continuar con el siguiente.

2. DESARROLLO DEL PROYECTO

2.1 Códigos de Goppa binarios

2.1.1 Definición de un código de Goppa

Los códigos de Goppa binarios son unos códigos lineales construidos sobre el cuerpo finito *GF(2) = {0,1}*. Para construir un código de Goppa se necesitan ciertos elementos principales:

- Un cuerpo finito auxiliar definido por un polinomio binario irreducible y primitivo (no es estrictamente necesario, pero es conveniente):

$$GF(2^m) = \{0, 1, \alpha, \alpha^2, \ \dots\ , \alpha^{2m-2}\}$$

- Un polinomio *g(z)* de grado *t* con coeficientes en $GF(2^m)$:

$$g(z) = g_0 + g_1 z + \dots + g_t z^t = \sum_{i=0}^{t} g_i z^i$$

- Un conjunto L de elementos de $GF(2^m)$. Este conjunto no puede contener ninguna raíz de *g(z)* $(g(\alpha_i) \neq 0)$:

$$L = \{\alpha_1, \dots, \alpha_n\} \subseteq GF(2^m)$$

Notaremos el código de Goppa determinado por estos elementos como: $\Gamma(L, g(z))$.

EJEMPLO:

- Cuerpo finito: $GF(2^4)$.
- Polinomio binario irreducible: $x^4 + x + 1$.
- Polinomio: $g(z) = (z + \alpha^{13})(z + \alpha^{14}) = z^2 + \alpha^2 z + \alpha^{12}$.
- Conjunto: $L = \{\alpha, \alpha^2, \alpha^3, \alpha^4, \alpha^5, \alpha^6, \alpha^7, \alpha^8, \alpha^9, \alpha^{10}, \alpha^{11}, \alpha^{12}\}$.

Un código de Goppa está formado por palabras binarias de n bits $c = (c_1, c_2, \dots, c_n)$ que cumplen:

$$R_c(z) = \sum_{i=1}^{n} \frac{c_i}{z - \alpha_i} \equiv 0 \,(mod\ g(z)) \qquad (1)$$

Esto puede parecer que, aparentemente, no tiene sentido, ya que se está dividiendo un bit (c_i) entre un polinomio. Lo que significa realmente es que se multiplica por el inverso del polinomio módulo $g(z)$. Veamos que esto tiene sentido. Como hemos mencionado antes, todos los elementos de L cumplen $g(\alpha_i) \neq 0$. Como $z - \alpha_i$ tiene grado 1, $mcd(z - \alpha_i\ ,\ g(z)) = 1$. Entonces, por la identidad de Bézout, existen dos polinomios tales que:

$$(z - \alpha_i) \cdot a_i(z) + g(z) \cdot b_i \equiv 1 \ (mod\ g(z)) \qquad (2)$$

Como el resultado es módulo *g(z)*, se omite la segunda parte de la ecuación y se obtiene:

$$(z - \alpha_i) \cdot a_i(z) \equiv 1 \ (mod\ g(z)) \qquad (3)$$

El polinomio $a_i(z)$ es el inverso del denominador de la ecuación (1):

$$a_i = \frac{1}{z - \alpha_i}$$

2.1.2 Codigos *separables*

Un polinomio binario de grado *t* es *separable* cuando no tiene raíces de multiplicidad más grande que 1. Esta condición da una mayor capacidad correctora y hace que se cumpla esta propiedad:

$$\Gamma(L, g(z)) = \Gamma(L, g^2(z)) \qquad (4)$$

No obstante, no se extiende a $\Gamma(L, g(z)) = \Gamma(L, g^4(z))$, ya que $g^2(z)$ ya no es *separable*.

2.1.3 Parámetros de los códigos de Goppa

Los parámetros principales de un código de Goppa son:

- *n*: el tamaño del código.

- k: la dimensión del código. La información a enviar se divide en bloques de k bits que, tras multiplicarse por una matriz generadora, pasarán a ser palabras del código de tamaño n. Satisface $k \geq n - mt$.
- d: distancia mínima (es decir, mínimo número de diferencias) entre las palabras del código. En general, $d \geq t + 1$. En cambio, si el código es *separable* y se usa un algoritmo de corrección adecuado, se satisface la condición $d \geq 2t + 1$.
- r: número de errores que se pueden corregir. Satisface $2r + 1 \leq d$.

Se suele usar la notación $[n, k, d]$ para indicar los parámetros de los códigos de Goppa.

2.1.4 Matriz de control de paridad

Se necesita una matriz de paridad H para decodificar. En general, una matriz de control es una matriz asociada a un sistema de ecuaciones lineales y homogéneas cuyas soluciones son las palabras del código. En nuestro caso, los elementos de esta matriz son los polinomios que anulan *c (mod g(z))*. Por lo tanto, podemos reescribir la ecuación (1) como:

$$\sum_{i=1}^{n} c_i p_{ij} = 0, \text{para } 1 \leq j \leq t$$

Como una palabra es del código solo si cumple esta igualdad, la matriz de paridad H satisface $cH^T = 0$. Así pues:

$$H = \begin{pmatrix} p_{11} & \cdots & p_{n1} \\ \vdots & \ddots & \cdots \\ p_{1t} & \cdots & p_{nt} \end{pmatrix} (5)$$

Los elementos de esta matriz son los polinomios $a_i(z)$ de la ecuación (3) del apartado *2.1.1*. Por ejemplo, si

$$\frac{1}{z - \alpha_1} \equiv \alpha^3 + \alpha^{11} z \ (mod\ g(z))$$

,

la primera columna de la matriz H será: $p_{11} = \alpha^3$, $p_{12} = \alpha^{11}$. Por lo tanto, para construir la matriz de paridad H se pueden buscar estos polinomios resolviendo la igualdad (2). Esto es bastante sencillo de aplicar a la hora de hacer una implementación a máquina, pero no tanto a mano. A continuación, se muestra otro método más sencillo para encontrar los elementos de H:

$$\begin{cases} p_{i1} = -\left(g_t \alpha_i^{t-1} + g_{t-1}\alpha_i^{t-2} + \cdots + g_2\alpha_i + g_1\right)h_i \\ p_{i2} = -\left(g_t \alpha_i^{t-2} + g_{t-1}\alpha_i^{t-3} + \cdots + g_2\right)h_i \\ \vdots \\ p_{i(t-1)} = -\left(g_t \alpha_i + g_{t-1}\right)h_i \\ p_{it} = -g_t h_i \end{cases} \qquad (6)$$

La variable h se define como $h_i := g(\alpha_i)^{-1}$.

2.1.5 Matriz generadora

Para codificar y decodificar mensajes se necesita una matriz generadora G. Una matriz generadora es una matriz cuyas filas forman una base del código. Por tanto, las palabras c del código se obtienen multiplicando un mensaje m arbitrario de longitud k por G. Una vez se ha calculado la matriz H, se puede obtener G a partir de la igualdad $GH^T = 0$. Esto se consigue resolviendo un sistema de ecuaciones.

2.1.6 Codificar y decodificar mensajes

Para codificar, se divide el mensaje a enviar en bloques de tamaño k y se multiplica por $G\,(k \times n)$:

$$(m_1, \ \ldots\ , m_k) \cdot G = (c_1, \ \ldots\ , c_n) \qquad (7)$$

Una vez hemos corregido los errores, para obtener el vector de bits de información (decodificar), se tiene que invertir la codificación anterior. A partir de (7) vemos que:

$$G^T \cdot \begin{pmatrix} m_1 \\ \vdots \\ m_k \end{pmatrix} = \begin{pmatrix} c_1 \\ \vdots \\ c_n \end{pmatrix}$$

Por lo tanto, podemos obtener *m* a partir de *c* resolviendo el sistema lineal anterior.

2.1.7 Corrección de errores

Se interpreta el vector *y* como la palabra del código enviada *c* que contiene *r* errores. Es decir:

$$(y_1, \ldots, y_n) = (c_1, \ldots, c_n) + (e_1, \ldots, e_n),$$

con $e_i \neq 0$ en *r* posiciones, correspondientes a las posiciones de error. En general,

$$r \leq \left\lfloor \frac{t}{2} \right\rfloor.$$

Aunque si el código es separable, y se usa un algoritmo de corrección adecuado, $2r + 1 \leq d$. Como son códigos binarios, el valor del error siempre es 1. Por lo tanto, para corregir una palabra, solo es necesario encontrar el conjunto de localización de errores $B = \{i \mid e_i \neq 0\}$.

Hay varios algoritmos eficientes para corregir errores. En el siguiente apartado se explicará uno de ellos.

Algoritmo para corregir errores

Este algoritmo en concreto solo corrige

$$r \leq \left\lfloor \frac{t}{2} \right\rfloor$$

errores. Pero, si es un código binario y *separable,* se cumple $\Gamma(L, g(z)) = \Gamma(L, g^2(z))$. Por lo tanto, $g^2(z)$ tiene grado $2t$ y el algoritmo puede corregir $\frac{2t}{2} = t$ errores. Ahora, la

matriz original H ya no es útil. Se tiene que encontrar una nueva matriz de paridad H' de $\Gamma(L, g^2(z))$.

Antes de pasar a explicar los pasos del algoritmo, introducimos $\sigma(z)$, el polinomio localizador de errores, como:

$$\sigma(z) := \prod_{i \in B} (z - \alpha_i)$$

Este polinomio se anula precisamente en aquellos elementos con índice i donde hay un error.

Algoritmo (*algoritmo 3.1* de [3] adaptado para códigos binarios *separables*):

1. *Computar el síndrome*

$$s(z) = \sum_{i=1}^{n} \frac{y_i}{z - \alpha_i}.$$

   Si el síndrome es 0, la palabra no contiene errores. Por lo tanto, es una palabra del código.

2. *Resolver la ecuación*

$$\sigma(z)s(z) \equiv \sigma'(z) \ (mod\ g(z)),$$

   donde $\sigma(z) = \sigma_0 + \sigma_1 z + ... + \sigma_{r-1} z^{r-1} + z^r$. *La prima indica la derivada del polinomio.*

3. *Determinar el conjunto de errores* $B = \{i \mid \sigma(\alpha_i) = 0\}$.

4. *Formar el vector de errores* $e = (e_1, \ldots, e_n)$ *con un uno en las posiciones determinadas por los elementos de B y un cero las restantes.*

5. *La palabra del código enviada es:* $c = y - e$.

2.1.8 Ejemplo de codificación, corrección y decodificación de un mensaje

Para este ejemplo vamos a utilizar el código de Goppa definido en el apartado *2.1.1*:

- Cuerpo finito auxiliar: $GF(2^4)$.
- Polinomio binario irreducible usado para construir el cuerpo finito: $x^4 + x + 1$.
- Polinomio: $g(z) = (z + \alpha^{13})(z + \alpha^{14}) = z^2 + \alpha^2 z + \alpha^{12}$.
- Conjunto: $L = \{\alpha, \alpha^2, \alpha^3, \alpha^4, \alpha^5, \alpha^6, \alpha^7, \alpha^8, \alpha^9, \alpha^{10}, \alpha^{11}, \alpha^{12}\}$.

Para encontrar el polinomio binario se puede factorizar $X^{15} - 1$ modulo 2. El polinomio escogido será primitivo sólo si el orden de α es 15. Por lo tanto, tenemos que comprobar, usando $\alpha^4 = \alpha + 1$, que $\alpha \neq 1$, $\alpha^3 \neq 1$, $\alpha^5 \neq 1$.

$$\alpha^5 = \alpha \cdot \alpha^4 = \alpha(\alpha + 1) = \alpha^2 + \alpha \neq 1$$

Ahora, representamos los elementos de $GF(2^4)$ como potencias de α usando que $\alpha^4 = \alpha + 1$.

$$\begin{aligned}
1 &= 1 &&= (1,0,0,0)^T \\
\alpha &= \alpha &&= (0,1,0,0)^T \\
\alpha^2 &= \alpha^2 &&= (0,0,1,0)^T \\
\alpha^3 &= \alpha^3 &&= (0,0,0,1)^T \\
\alpha^4 &= 1 + \alpha &&= (1,1,0,0)^T \\
\alpha^5 &= \alpha + \alpha^2 &&= (0,1,1,0)^T \\
\alpha^6 &= \alpha^2 + \alpha^3 &&= (0,0,1,1)^T \\
\alpha^7 &= 1 + \alpha + \alpha^3 &&= (1,1,0,1)^T \\
\alpha^8 &= 1 + \alpha^2 &&= (1,0,1,0)^T \\
\alpha^9 &= \alpha + \alpha^3 &&= (0,1,0,1)^T \\
\alpha^{10} &= 1 + \alpha + \alpha^2 &&= (1,1,1,0)^T \\
\alpha^{11} &= \alpha + \alpha^2 + \alpha^3 &&= (0,1,1,1)^T \\
\alpha^{12} &= 1 + \alpha + \alpha^2 + \alpha^3 &&= (1,1,1,1)^T \\
\alpha^{13} &= 1 + \alpha^2 + \alpha^3 &&= (1,0,1,1)^T \\
\alpha^{14} &= 1 + \alpha^3 &&= (1,0,0,1)^T
\end{aligned} \qquad (8)$$

Los parametros de este código son: $m = 4$, $n = 12$, $t = 2$, $k \geq 4$ ($k \geq n - mt$), $d \geq 5$ ($2t + 1$). Por lo tanto, es un código de Goppa con parámetros $[12, \geq 4, \geq 5]$.

Vemos que, $\alpha_1 = \alpha$, $\alpha_2 = \alpha^2$, ... , $\alpha_{12} = \alpha^{12}$ y $g_0 = \alpha^{12}$, $g_1 = \alpha^2$, $g_2 = 1$. Ahora pasamos a calcular la matriz de paridad H (5). Primero debemos calcular $h_i := g(\alpha_i)^{-1}$:

$$h_1 = g(\alpha)^{-1} = ((\alpha)^2 + \alpha^2 \cdot \alpha + \alpha^{12})^{-1} = (\alpha^2 + \alpha^3 + \alpha^{12})^{-1}$$
$$h_1 = ((0,0,1,0)^T + (0,0,0,1)^T + (1,1,1,1)^T)^{-1}$$
$$h_1 = ((1,1,0,0)^T)^{-1} = \alpha^{-4} = \alpha^{11}.$$

$$h_2 = g(\alpha^2)^{-1} = ((\alpha^2)^2 + \alpha^2 \cdot \alpha^2 + \alpha^{12})^{-1} = (\alpha^4 + \alpha^4 + \alpha^{12})^{-1}$$
$$h_2 = ((1,1,0,0)^T + (1,1,0,0)^T + (1,1,1,1)^T)^{-1}$$
$$h_2 = ((1,1,1,1)^T)^{-1} = \alpha^{-12} = \alpha^3.$$

Una vez calculadas todas las h_i, calculamos los valores de la matriz H usando (6):

$$\begin{cases} p_{i1} = -\left(g_2 \alpha_i + g_1\right) \cdot h_i \\ p_{i2} = -g_2 \cdot h_i \end{cases}$$

Por ejemplo, $p_{11} = -(\alpha + \alpha^2) \cdot \alpha^{11} = \alpha^{16} = \alpha$, $p_{12} = \alpha^{11}$. Después de algunos cálculos, obtenemos:

$$H = \begin{pmatrix} \alpha & 0 & \alpha^{13} & \alpha^5 & \alpha^{12} & \alpha^{10} & \alpha^6 & \alpha^6 & \alpha^{12} & \alpha^{14} & \alpha^{10} & \alpha \\ \alpha^{11} & \alpha^3 & \alpha^7 & \alpha^{10} & \alpha^{11} & \alpha^7 & \alpha^9 & \alpha^6 & \alpha & \alpha^{10} & \alpha & \alpha^9 \end{pmatrix}$$

Podemos verificar que la matriz H es correcta si las columnas son iguales a los polinomios de (1) *(mod g(z))*. Por ejemplo, verificamos la última columna:

$$1 \div (z - \alpha^{12}) \equiv \alpha + \alpha^9 z \ (mod\ g(z))$$

$$\begin{aligned}(z-\alpha^{12})(\alpha+\alpha^9 z) &= \alpha^9 z^2+(\alpha+\alpha^6)z+\alpha^{13}\\ &\equiv \alpha^9 z^2+\alpha^{11}z+\alpha^{13}+\alpha^9(z^2+\alpha^2 z+\alpha^{12})\\ &= \alpha^{11}z+\alpha^{13}+\alpha^{11}z+\alpha^6\\ &= \alpha^{13}+\alpha^6\\ &= 1\end{aligned}$$

Ahora, expresamos la matriz H de forma binaria usando la tabla (8).

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

La matriz G se puede encontrar a través de H usando $GH^T = 0$. Esto puede resolverse encontrando las soluciones del sistema homogéneo con matriz asociada H (módulo 2). Una base del espacio de las soluciones viene dada por las filas de G.

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

Se puede apreciar que la dimensión del código es 4. Por lo tanto, es un código de Goppa $[12, 4, \geq 5]$.

Ya tenemos todo lo necesario para codificar, corregir errores y decodificar. Codificamos, por ejemplo, el mensaje $m = (1, 1, 0, 0)$.

$$c = (1,1,0,0)\cdot\begin{pmatrix} 1&0&0&1&1&1&0&1&0&0&0&0\\ 1&1&0&0&0&1&1&0&0&1&0&0\\ 0&1&1&1&1&0&1&0&1&0&1&0\\ 1&1&0&0&1&0&0&0&1&0&0&1 \end{pmatrix}$$
$$= (0,1,0,1,1,0,1,1,0,1,0,0)$$

Como el $t = 2$ y $d \geq 5$, el código puede corregir $r \leq 2$ errores. Añadimos errores en las posiciones *4* y *5*.

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

Ahora, pasamos a aplicar el algoritmo de corrección de errores. Pero, para poder corregir hasta *t* errores con el algoritmo explicado en el apartado anterior (*2.1.7*), es necesario usar el polinomio generador $g'(z) = g^2(z)$. Como este código es *separable*, vemos que $\Gamma(L, g(z)) = \Gamma(L, g^2(z))$. Por lo tanto, debemos calcular $g^2(z)$ y su matriz de paridad correspondiente H'.

$$g'(z) = g^2(z) = (z^2 + \alpha^2 z + \alpha^{12})^2 = z^4 + \alpha^4 z^2 + \alpha^9$$

Observamos que ahora, $g_4 = 1,\ g_3 = 0,\ g_2 = \alpha^4,\ g_1 = 0,\ g_0 = \alpha^9,$ y $h'_i = h_i^2$. Calculamos los valores de *H* usando (6):

$$\begin{cases} p_{i1} = -\left(g_4\alpha_i^3 + g_3\alpha_i^2 + g_2\alpha_i + g_1\right)h_i^2 \\ p_{i2} = -\left(g_4\alpha_i^2 + g_3\alpha_i + g_2\right)h_i^2 \\ p_{i3} = -\left(g_4\alpha_i + g_3\right)h_i^2 \\ p_{i4} = -g_4 h_i^2 \end{cases}$$

Por ejemplo, la primera columna sería:

- $p_{11} = (\alpha^3 + (\alpha^4 \cdot \alpha)) \cdot (\alpha^{11})^2 = \alpha^{11} \cdot \alpha^7 = \alpha^3$
- $p_{12} = (\alpha^2 + \alpha^4) \cdot (\alpha^{11})^2 = \alpha^{10} \cdot \alpha^7 = \alpha^2$
- $p_{13} = \alpha \cdot (\alpha^{11})^2 = \alpha^8$
- $p_{14} = (\alpha^{11})^2 = \alpha^7$

Después de calcular todas los valores, obtenemos

$$H' = \begin{pmatrix} \alpha^3 & 0 & \alpha^{14} & \alpha^{14} & \alpha^{14} & \alpha^{11} & \alpha^4 & \alpha^5 & \alpha^3 & \alpha^8 & \alpha & \alpha^{14} \\ \alpha^2 & 0 & \alpha^{11} & \alpha^{10} & \alpha^9 & \alpha^5 & \alpha^{12} & \alpha^{12} & \alpha^9 & \alpha^{13} & \alpha^5 & \alpha^2 \\ \alpha^8 & \alpha^8 & \alpha^2 & \alpha^9 & \alpha^{12} & \alpha^5 & \alpha^{10} & \alpha^5 & \alpha^{11} & 1 & \alpha^{13} & 1 \\ \alpha^7 & \alpha^6 & \alpha^{14} & \alpha^5 & \alpha^7 & \alpha^{14} & \alpha^3 & \alpha^{12} & \alpha^2 & \alpha^5 & \alpha^2 & \alpha^3 \end{pmatrix}$$

Podemos verificar las columnas de la misma manera que lo hemos hecho antes con H.

Ahora, que ya tenemos $g'(z)$ y H', podemos aplicar el algoritmo de corrección a la palabra $y = (0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0,)$:

1. Calculamos el síndrome:

$$s(z) = \sum_{i=1}^{n} \frac{y_i}{z - \alpha_i} = \frac{1}{z - \alpha^2} + \frac{1}{z - \alpha^7} + \frac{1}{z - \alpha^8} + \frac{1}{z - \alpha^{10}}$$
$$\equiv \left(\alpha^4 + \alpha^5 + \alpha^8\right) + \left(\alpha^{12} + \alpha^{12} + \alpha^{13}\right)z + \left(\alpha^8 + \alpha^{10} + \alpha^5 + 1\right)z^2$$
$$+ \left(\alpha^6 + \alpha^3 + \alpha^{12} + \alpha^5\right)z^3 = \alpha^{13}z^3 + \alpha^8 z^2 + \alpha^{13} z$$

2. Calculamos $\sigma(z)s(z)$ $(mod\ z^4 + \alpha^4 z^2 + \alpha^9)$

$$\begin{aligned} \sigma(z)s(z) &= \left(z^2 + \sigma_1 z + \sigma_0\right)\left(\alpha^{13}z^3 + \alpha^8 z^2 + \alpha^{13} z\right) \\ &= \alpha^{13}z^5 + \left(\alpha^{13}\sigma_1 + \alpha^8\right)z^4 + \left(\alpha^{13} + \alpha^8\sigma_1 + \alpha^{13}\sigma_0\right)z^3 + \\ &\quad \left(\alpha^{13}\sigma_1 + \alpha^8\sigma_0\right)z^2 + \alpha^{13}\sigma_0 z \\ &\equiv \left(\alpha^{13}\sigma_1 + \alpha^8\right)z^4 + \left(\alpha^{13} + \alpha^8\sigma_1 + \alpha^{13}\sigma_0 + \alpha^2\right)z^3 + \\ &\quad \left(\alpha^{13}\sigma_1 + \alpha^8\sigma_0\right)z^2 + \left(\alpha^{13}\sigma_0 + \alpha^7\right)z \\ &\equiv \left(\alpha^{13} + \alpha^8\sigma_1 + \alpha^{13}\sigma_0 + \alpha^2\right)z^3 + \left(\alpha^{13}\sigma_1 + \alpha^8\sigma_0 + \alpha^2\sigma_1 + \alpha^{12}\right)z^2 + \\ &\quad \left(\alpha^{13}\sigma_0 + \alpha^7\right)z + \left(\alpha^7\sigma_1 + \alpha^2\right) \end{aligned}$$

En el tercer paso, le hemos añadido $\alpha^{13} z\, g'(z)$ y en el cuarto paso, $\alpha^{13}\sigma_1 + \alpha^8\, g'(z)$.

A continuación, resolvemos la ecuación $\sigma(z)s(z) \equiv \sigma'(z)$ $(mod\ z^4 + \alpha^4 z^2 + \alpha^9)$, donde $\sigma'(z) = \sigma_1$ (ya que $2 \equiv 0$ módulo 2) mediante un sistema de ecuaciones:

$$\begin{cases} \sigma_1 &= \alpha^7\sigma_1 + \alpha^2 \\ 0 &= \alpha^{13}\sigma_0 + \alpha^7 \\ 0 &= \alpha^{13}\sigma_1 + \alpha^8\sigma_0 + \alpha^2\sigma_1 + \alpha^{12} \\ 0 &= \alpha^{13} + \alpha^8\sigma_1 + \alpha^{13}\sigma_0 + \alpha^2 \end{cases}$$

Obtenemos $\sigma_0 = \alpha^9$ y $\sigma_1 = \alpha^8$, por lo tanto

$$\sigma(z) = z^2 + \alpha^8 z + \alpha^9 = (z + \alpha^4)(z + \alpha^5).$$

3. Como en este código $\alpha_4 = \alpha^4$ y $\alpha_5 = \alpha^5$, el conjunto de errores es $B = \{4, 5\}$.
4. El vector de errores es $e = (0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0)$.
5. La palabra del código es

$$c = y - e = (0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0,).$$

Observamos que hemos corregido el error correctamente.

Ahora, ya podemos pasar a decodificar para encontrar el mensaje original *m*:

$$\left[G^T \middle| c^T\right] = \left(\begin{array}{cccc|c} 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{array}\right) \sim \left(\begin{array}{cccc|c} 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ \hline 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \end{array}\right)$$

Ya hemos recuperado la información transmitida $m = (1, 1, 0, 0)$.

2.2 Criptosistema de McEliece

2.2.1 Análisis del sistema de McEliece

Para crear un criptosistema basado en un código de Goppa son necesarios todos los ingredientes del código: un polinomio arbitrario (a poder ser *separable)* $g(z)$ de grado t sobre $GF(2^m)$ y el conjunto de elementos L. Una vez obtenido un código de Goppa $\Gamma(L, g(z))$ $[n, k, d]$, se computa la matriz generadora G $(k \times n)$. Entonces, ya es posible la generación de las claves.

La clave privada, descrita como $\{G, S, P\}$, está formada por tres matrizes de elementos binarios:

- G, la matriz generadora de tamaño $k \times n$.
- S, una matriz $k \times k$ invertible seleccionada aleatoriamente.
- P, una matriz $n \times n$ de permutación seleccionada aleatoriamente.

La clave pública, que corresponde a la clave privada $\{G, S, P\}$, es el par $\{G', t\}$ definido como:

- $G' = SGP$, una matriz de tamaño $k \times n$.
- t, es el grado del polinomio $g(z)$.

2.2.2 Encriptar y desencriptar un mensaje

2.2.2.1 Encriptar

Para encriptar la información, se divide el mensaje en bloques de tamaño k, se multiplica por G' y se añaden errores aleatorios (t errores como máximo en el caso *separable*). En resumen:

$$y = mG' + e = m(SGP) + e.$$

2.2.2.2 Desencriptar

Puede dividirse en cuatro sencillos pasos:

1. Se multiplica y por la inversa de P: $y' = yP^{-1} = (mS)G + eP^{-1}$.
2. Se corrigen los errores de y'. Como P es una matriz de permutación, el algoritmo de corrección se puede usar ya que $|eP^{-1}| = |e| = t$.[4] Una vez encontrados los errores e' de y', se obtiene $c' = (mS)G$.
3. Se resuelve el sistema de ecuaciones como en el apartado *2.1.6.2* y se obtiene $m' = mS$.
4. Se calcula $m = m'S^{-1}$, y se obtiene el mensaje m original.

2.2.3 Ejemplo del sistema de McEliece

Para este ejemplo usaremos el código de Goppa construido en el apartado *2.1.8*. Recordemos que es un código de Goppa $[12, 4, \geq 5]$. Primero, debemos formar la clave privada $\{G, S, P\}$. La matriz generadora

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

ya la tenemos. Solo debemos escoger una matriz invertible S aleatória y una matriz P de permutación aleatoria. Por ejemplo:

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

[4] |e| indica el peso del vector e, es decir, el número de posiciones no nulas.

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

Ya tenemos la clave privada. Ahora, necesitamos G' para formar la clave pública:

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

Ahora ya es posible encriptar mensajes. Suponemos que queremos encriptar el mensaje $m = (0, 1, 0, 1)$. Primero, calculamos

$$mG' = (0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 0).$$

Ahora, añadimos un conjunto de errores ($\leq r = 2$). Escogemos, por ejemplo, el conjunto de las dos últimas posiciones. Entonces, enviamos el mensaje *y*:

$$y = mG' + e = (0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1)$$

Si el receptor posee la clave privada, es capaz de desencriptar el mensaje aplicando los cuatro pasos explicados en el apartado *2.2.2.2*:

1. Calculamos: $y' = yP^{-1} = (mS)G + eP^{-1} = (0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 0).$

2. Ahora corregimos el error de la nueva palabra y' con un algoritmo de corrección de errores y obtenemos:

$$c' = (mS)G = (1,0,0,1,1,1,0,1,0,0,0,0)$$

Los errores de esta palabra del código estaban en las posiciones uno y once.

3. A continuación, resolvemos el sistema de ecuaciones para decodificar la palabra c' y obtenemos $m' = mS$:

$$\left[G^T \middle| c'^T\right] = \left(\begin{array}{cccc|c} 1&1&0&1&1\\ 0&1&1&1&0\\ 0&0&1&0&0\\ 1&0&1&0&1\\ 1&0&1&1&1\\ 1&1&0&0&1\\ 0&1&1&0&0\\ 1&0&0&0&1\\ 0&0&1&1&0\\ 0&1&0&0&0\\ 0&0&1&0&0\\ 0&0&0&1&0 \end{array}\right) \sim \left(\begin{array}{cccc|c} 1&0&0&0&1\\ 0&1&0&0&0\\ 0&0&1&0&0\\ 0&0&0&1&0\\ \hline 1&1&0&1&1\\ 0&1&1&1&0\\ 0&0&1&0&0\\ 1&0&1&0&1\\ 1&0&1&1&1\\ 1&1&0&0&1\\ 0&1&1&0&0\\ 0&0&1&1&0 \end{array}\right)$$

Por lo tanto, $m' = (1,0,0,0)$.

4. Para concluir, calculamos $m = m'S^{-1} = (0,1,0,1)$. Como podemos observar, hemos recuperado el mensaje original correctamente.

2.2.4 Seguridad del sistema de McEliece

Los valores iniciales que propuso McEliece para ilustrar diversos aspectos de su sistema fueron: $n = 1024$, $k = 524$ y $t = 50$. Estos valores parecen bastante seguros si se intenta atacar al sistema con fuerza bruta, pero hay ataques más inteligentes.

2.2.4.1 Fuerza bruta

El problema de decodificar códigos lineales es un problema NP-completo [27].

Una de las opciones es generar las 2^k palabras del código (de G') y buscar la palabra más cercana a y. Entonces decodificarla resolviendo $c = mG'$. Con los parámetros propuestos por McEliece esto requiere un cálculo de

$$2^k = 2^{524} = 5'491838 \cdot 10^{157}$$

comparaciones. Esto es suficientemente seguro.

Intentar encontrar S y P de forma aleatória es aún más costoso. La probabilidad es ínfimamente pequeña. Por ejemplo, hay 1024! matrices de permutación de orden 1024. Por lo tanto, si se pretende atacar al sistema, es conveniente usar un método más eficiente.

2.2.4.2 Seleccionar *k* coordenadas sin error

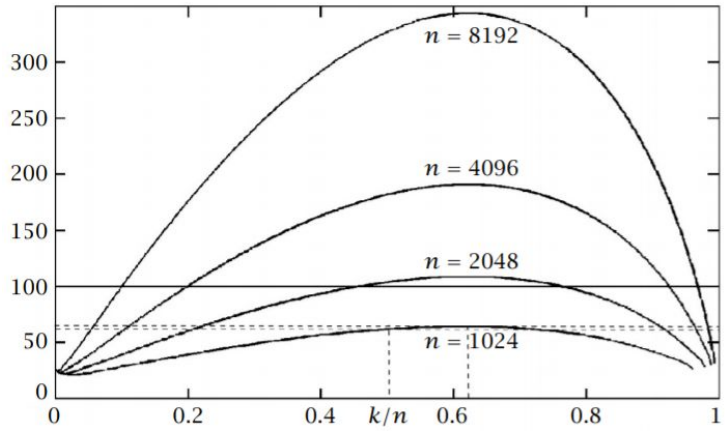
El ataque conocido más efectivo contra este criptosistema consiste en intentar decodificar un conjunto de la información. En este apartado se va a explicar la forma básica de este ataque, que fue introducida por McEliece [1]. No obstante, más adelante, han surgido modificaciones del método. Concretamente, las variantes introducidas por Leon en [6], por Lee y Brickell en [7], por Stern en [8], por van Tilburg en [9], por Canteaut and Chabanne en [10], por Canteaut and Chabaud en [11], y por Canteaut and Sendrier en [12].

Este ataque consiste en seleccionar k coordenadas de y aleatoriamente y esperar que no contenga errores. Si el conjunto seleccionado no contiene ningún error, es posible obtener m resolviendo el sistema de ecuaciones $\overline{y} = m\overline{G}$, donde $\overline{y}$ es el vector seleccionado de y y $\overline{G}$ es la submatriz de G con las columnas correspondientes a los índices de $\overline{y}$.

El método de Gauss para resolver un sistema de ecuaciones lineales con k ecuaciones requiere del orden de k^3 operaciones y la probabilidad de que las k posiciones sean correctas es aproximadamente $(1 - k/n)^k$. Por lo tanto, para los parámetros escogidos por McEliece, se deben realizar alrededor de

$$\frac{k^3}{\left(1-\frac{t}{n}\right)^k}=\frac{524^3}{\left(1-\frac{50}{1024}\right)^{524}}=3'5504036864\cdot 10^{19}$$

operaciones. Como este número es superior a 2^{64}, el sistema llega a una seguridad de 64 bits (si se ataca con este método).



Esta figura es una adaptación de la figura del apartado 6.2 de [5] expuesta en el artículo [2]. Representa el grado de seguridad del sistema de McEliece, frente a este tipo de ataque, en función de $R = k/n$ para diferentes longitudes n del código. El índice izquierdo de la figura representa el nivel de seguridad, en función de logaritmo en base 2, del ataque más fuerte conocido. En esta figura se puede apreciar la importancia del ratio $R = k/n$ del código. La línea discontinua inferior indica el grado de seguridad, de 63 bits, con los parámetros propuestos por McEliece (con un ratio $R = 524/1024 = 0.512$). La superior indica el ratio R más seguro, 64 bits, con $n = 1024$. Por lo tanto, se puede observar que el ratio R más seguro es *0.625*.

2.3 Propuesta del NIST

Como hemos mencionado en la introducción, el NIST ha abierto un concurso público para buscar un criptosistema capaz de soportar ataques de computadores cuánticos. De entre todas las propuestas presentadas, hay una que parece ser bastante prometedora basada precisamente en el criptosistema clásico de McEliece explicado en este proyecto. Por lo tanto, en este apartado explicaremos algunos puntos importantes de su trabajo y los resultados que han obtenido sin entrar demasiado en detalle.

La propuesta se llama "*Classic McEliece: conservative code-based cryptography"* [13].

Uno de los objetivos principales del estudio de la propuesta es mantener el alto grado de seguridad del criptosistema original (que se ha puesto a prueba durante décadas) incluso después de hacer pequeñas modificaciones (explicadas con detalle en [13]).

Durante este proyecto hemos visto la seguridad del criptosistema en sí (en cuanto a la decodificación), pero también es esencial un alto nivel de seguridad en la encapsulación de paquetes, intercambio de llaves, confirmaciones, etc. En el estudio de la propuesta se tienen muy en cuenta estos factores.

2.3.1 Conceptos generales

2.3.1.1 Funcionamiento del criptosistema

El criptosistema que usan es una adaptación del criptosistema de Niederreiter, que es un muy similar al de McEliece. La principal diferencia entre los dos es que el criptosistema de Niederreiter usa como matriz generadora una matriz de paridad *H*.

La matriz generadora/de paridad que usan en la propuesta es una matriz $H' = \{h_{i,j}\}$ de tamaño $t \times n$, donde

$$h_{i,j} = \alpha_j^{i-1}/g(\alpha_j) \text{ para } i = 1, \ldots t \text{ y } j = 1, \ldots, n.$$

Ahora, pasamos H' a una matriz $mt \times n$ donde cada elemento pasa a ser los elementos en función de la potencia de *z* (como las matrices de paridad vistas durante todo el proyecto). Después, esta matriz de paridad H' se reduce a la forma $(I_{n-k} \mid T)$, donde I_{n-k} es la matriz de identidad $(n-k) \times (n-k)$ y T es la matriz sobrante. Si esta reducción no es posible, se debe buscar un nuevo código de Goppa Γ (cambiando el polinomio generador, los elementos de *L,* o ambos). Para terminar, solo necesitamos una tira *s* uniformemente aleatória de *n* bits.

Una vez tenemos todos estos elementos, ya podemos formar las claves. La clave privada es el par (s, Γ) y la clave pública es la matriz *T*.

No es necesario explicar los métodos de codificación, decodificación, encapsulación y decapsulación, para entender los resultados del estudio de la propuesta. Se pueden encontrar en [13, capítulo 2].

2.3.1.2 Modificación de la matriz generadora

Como hemos mencionado en el apartado anterior, la matriz generadora es una reducción de la matriz de paridad. El problema es que solo aproximadamente el 29% de los distintos Γ tienen una matriz H' capaz de reducirse a la forma $(I_{n-k} \mid T)$. Esto quiere decir que la generación de la clave será alrededor de 3'4 veces más lenta en media. También debemos tener en cuenta que, al limitar el número de códigos a un 29%, se reduce la seguridad en al menos 2 bits.

2.3.1.3 Parametros escogidos para poner a prueba

- kem/mceliece6960119: $m = 13$, $n = 6960$, $t = 119$. Polinomio que define el cuerpo finito: $f(z) = z^{13} + z^4 + z^3 + z + 1$.

- kem/mceliece8192128: $m = 13$, $n = 8192$, $t = 128$. Polinomio que define el cuerpo finito: $f(z) = z^{13} + z^4 + z^3 + z + 1$.

2.3.2 Seguridad previa

Como ya se ha mencionado varias veces, a mayor tamaño de parámetros mayor seguridad (y mayor tamaño de clave). El criptosistema de McEliece usa una clave de tamaño $(c_0 + o(1))b^2(lg\ b)^2$ para lograr una seguridad de 2^b bits teniendo en cuenta todos los ataques conocidos hasta 1978. lg es logaritmo en base *2*, $o(1)$ es algo que convergue a *0* a medida que b tiene a infinito, y $c_0 = R/(1-R)(lg(1-R))^2$ (su valor minimo es aproximadamente $0'7418860694$). Recordamos $R = k/n$.

Esto es aplicable a una computación cuántica. Sin embargo, aplicando el algoritmo de Grover [14], se reduce el coste del ataque a su raíz cuadrada asintóticamente si nos enfrentamos a un ataque cuántico. En este caso, el tamaño de la llave sería de $(4c_0 + o(1))b^2(lg\ b)^2$ bits. También se debe mencionar que el valor de $o(1)$ no es fijo y varía a medida que se perfeccionan los métodos de ataque.

2.3.3 Análisis del rendimiento obtenido

2.3.3.1 Tiempo

- mceliece8192128 software:
    - Encapsulación: algo menos de 300.000 ciclos.
    - Decapsulación: algo mas de 458.340 ciclos.
    - Generación de la llave: aproximadamente 2 billones de ciclos por intento. Se suelen necesitar 2-3 intentos.

- mceliece8192128 hardware:
    - Generación de llave: 1.173.750 ciclos por intento.
    - Decodificación: 17.140 ciclos.

- mceliece6960119 hardware:
    - Generación de llave: 966.400 ciclos por intento.
    - Decodificación: 17.055 ciclos.

2.3.3.2 Tamaño

- mceliece8192128:
    - Llave pública: 1.357.824 bytes.
    - Llave privada: 14.080 bytes.
    - Texto cifrado: 240 bytes.
    - Llave de sesión: 32 bytes.

- mceliece6960119:
    - Llave pública: 1.047.319 bytes.
    - Llave privada: 13.908 bytes.
    - Texto cifrado: 226 bytes.
    - Llave de sesión: 32 bytes.

2.3.3.3 Como afectan los parámetros al rendimiento

El tamaño de los textos cifrados (bloques de información) es de $n - k$ bits. El ratio $R = n/k$ suele ser aproximadamente *0'8*; por lo tanto, el tamaño de los textos cifrados es

aproximadamente $0'2n$ ($n/40$ bytes). Aunque también se le añaden 32 bytes de confirmación.

El tamaño de la clave pública es $k(n-k)$ bits. Para un ratio R de 0'8, son aproximadamente $0'16n^2$ bits ($n^2/50$ bytes).

Generar la clave pública requere $n^{3+o(1)}$ operaciones con la eliminación Gaussiana estándar. Sin embargo, hay algoritmos más rápidos asintóticamente. La clave privada requiere solo $n^{1+o(1)}$ operaciones con algoritmos estándar.

2.3.4 De la teoría a la práctica

La fuerza esperada para los parámetros propuestos es de IND-CAA2[5] KEM Categoría 5.

La propuesta pretende conseguir esta seguridad a travé de los siguientes cuatro puntos:

- El criptosistema de McEliece es un sistema de clave pública OW-CPA[6].
- El criptosistema de Niederreiter está basado en el de McEliece y también adquiere la propiedad OW-CPA.
- Usar un sistema de encapsulación que adquiera una seguridad IND-CCA2.
- Preservar esta seguridad IND-CCA2 a pesar de las pequeñas modificaciones de la propuesta.

2.3.5 Análisis de ataques conocidos

2.3.5.1 Ataque de decodificación del conjunto de información

En el apartado *2.2.4.2 (Seleccionar k coordenadas sin error)* se habla sobre este tipo de ataques; y como ya hemos mencionado, esta es la forma conocida más eficiente de atacar al criptosistema.

[5] La indistinguibilidad de un texto cifrado es una propiedad que tienen muchos criptosistemas. Si se cumple esta propiedad, un atacante no debe poder romper el criptosistema a partir de un texto que él mismo encripta. Un criptosistema es IND-CCA seguro cuando el atacante no tiene prácticamente ninguna ventaja para romperlo. IND-CCA2 es el grado de seguridad más fuerte.

[6] Un criptosistema es OW-CPA ("one-way", sentido único) cuando un atacante no puede encontrar de manera eficiente las palabras del código a partir de un texto cifrado con la clave pública.

Los parametros de mceliece6960119 ($m = 13,\ n = 6960,\ t = 119$) fueron propuestos en un paper [16] que rompió los parámetros originales de McEliece (10, 1024, 50).

El paper informa que su ataque usa $2^{266'94}$ operaciones de bits para romper los parámetros (13, 6960, 119). Sin embargo, las mejoras posteriores de este tipo de ataques reducen el número de operaciones de bits por debajo de 2^{256}.

2.3.5.2 Recuperación de la clave

Otro tipo de ataque es intentar encontrar la clave privada. Es decir, los elementos que forma el código de Goppa $\Gamma(g, \alpha_1, \ldots, \alpha_n)$. Pero no necesariamente usando la fuerza bruta. Se puede obtener la secuencia $(\alpha_1, \ldots, \alpha_n)$ a partir de g y el conjunto $\{\alpha_1, \ldots, \alpha_n\}$; o determinar g a partir de $(\alpha_1, \ldots, \alpha_n)$.

Para mceliece6960119, el número de opciones para g es mayor que 2^{1500}; y para mceliece8192128, mayor que 2^{1600}. Aunque hay ciertas simetrías conocidas, sólo proporcionan una pequeña mejora. El número de opciones de $(\alpha_1, \ldots, \alpha_n)$ es mucho mayor. En este caso, mceliece6960119 tiene mayor defensa ya que tiene muchas más combinaciones posibles del conjunto $\{\alpha_1, \ldots, \alpha_n\}$.

2.3.5.3 Forzar pequeños errores

Otra manera de atacar el sistema es añadiendo errores. Por ejemplo, añadiendo dos errores al texto cifrado. Como la decodificación tiene éxito sólo si el número de errores es exactamente t, el atacante puede llegar a encontrar todos los errores analizando los fallos en la decodificación; ya que si se añaden dos errores y se decodifica con éxito, uno de los dos errores es correcto.

Sin embargo, este ataque no funciona en el criptosistema de la propuesta por dos motivos. Primero, el método de decapsulación fuerza a incluir un hash del vector de errores en la confirmación, y no es posible conseguir un hash modificado del vector sin saber el vector original. Y segundo, no es posible saber si la decodificación a fallado; ya que el texto modificado producirá una llave de sesión que no revela si el vector de errores tiene peso t.

2.3.6 Conclusiones de la propuesta

La principal ventaja de este criptosistema es el alto grado de seguridad. Está basado en un criptosistema que se ha estudiado y atacado durante más de 40 años. El objetivo principal de la propuesta es preservar la seguridad a pesar de las modificaciones realizadas.

En cuanto a la eficiencia, hay dos puntos principales a valorar:

- Tamaño: el tamaño de la clave privada es demasiado grande, ya que es una matriz completa; y es principalmente por este motivo, que el criptosistema apenas ha recibido uso. Sin embargo, el tamaño de los textos cifrados es considerablemente pequeño (256 bytes en su parámetro más grande). Esto permite que los textos cifrados puedan introducirse en un solo paquete de red.

- Tiempo: Aquí tenemos un problema similar. El tiempo de generación de llaves es bastante grande (sobretodo en software). No obstante, la encapsulación y la decapsulación son razonablemente rápidas; ya que usan operaciones simples con objetos binarios. Además, la decodificación también tiene un tiempo muy pequeño en hardware.

Como conclusión final, podemos observar que este criptosistema tendrá una eficiencia u otra dependiendo del uso que reciba. Transmitir textos cifrados pequeños puede ser más importante para el tráfico total que el gran tamaño de la llave. Por lo tanto, el objetivo para conseguir una mejor eficiencia es usar la misma llave tantas veces como sea posible; ya que el problema radica en el tamaño y transmisión de la llave pública.

3. PLANIFICACIÓN DEL PROYECTO

3.1 Plan de acción y riesgos

3.1.1 Turno de lectura

Según la página web oficial de la Facultad de Informática de Barcelona los Trabajos de Final de Grado podrán presentarse en dos turnos distintos: a finales de enero o a finales de abril. Considerando el gran volumen de trabajo y las fechas previstas para terminar este proyecto, se ha optado por el segundo turno (semana del 23 al 26 de abril del 2019).

Este proyecto se ha tenido que aplazar un quadrimestre por motivos personales del autor. Por lo tanto, las fechas han sido actualizadas. Sin embargo, no ha surgido ningún imprevisto en cuanto al proyecto.

3.1.2 Riesgos y alternativas

No se prevén demasiados riesgos para este proyecto ya que es un proyecto bastante teórico. Se cree bastante posible realizar la entrega en la fecha señalada. Estos serían tres de los posibles imprevistos que podrían surgir (todos ellos riesgos leves y poco probables):

1. Problemas con el apartado de computación cuántica (apartado *2.3*): Estamos acostumbrados a trabajar y pensar los algoritmos con una computación clásica; y la computación cuántica es un tema muy extenso y complejo. Esto podría dificultar este apartado y ocupar más tiempo de lo previsto. La solución simplemente sería dedicar unas horas extra; y si fuera necesario, pedir algo de ayuda a algún profesor de la asignatura de computación cuántica.

2. Problemas al entender a fondo la propuesta a estudiar: al ser una propuesta muy nueva y estar "rozando" el estado del arte, no hay mucha documentación sobre ello, y puede dar lugar a algún problema de comprensión. Pero los autores de la propuesta ya saben que es un tema muy reciente y no se espera problemas graves. Además, ocupará una gran parte del proyecto y pretende ampliarse todo lo posible; por lo tanto hay horas suficientes dedicadas a este apartado para evitar no llegar al plazo estimado.

3. Problemas con la implementación: La implementación del criptosistema de la propuesta está pensado para una computación cuántica. Por lo tanto, se tiene pensado hacer una implementación adaptada (con parámetros más realistas) para poder ponerla a prueba con una computación tradicional. Si se adapta bien, se cree que será bastante viable hacerla correctamente.

3.2 Recursos

3.2.1 Recursos personales

Como ya se ha mencionado anteriormente, solo se dispone de un trabajador. Dicho trabajador realizará los cuatro apartados iterativamente con la ayuda de reuniones semanales con el director del proyecto. El tiempo estimado del trabajador al proyecto será de unas 20-25 horas semanales.

3.2.2 Recursos materiales

- Ordenador portatil Lenovo: ordenador donde el trabajador desarrollará el proyecto. Se usará tanto para la escribir la memoria del proyecto como para escribir el código de la implementación.

- Documentos de google: se usará esta herramienta para escribir la memoria por comodidad; ya que es sencillo y rápido de usar, el documento permanece siempre en la “nube”, y se puede compartir con el director para que pueda comprobar que todo va como es debido (poner poner comentarios sobre el texto e incluso corregir en el mismo documento). Se usará, probablemente, otro editor de texto para la versión definitiva de la memoria una vez esté terminada.

- Correo electrónico de la FIB: herramienta de comunicación con el director del proyecto.

- SageMath: herramienta que se usará para realizar la implementación del código. Es un sistema algebraico computacional (CAS en inglés) muy útil en la algoritmia y matemáticas; ya que tiene paquetes y funciones predefinidas. Tiene funciones muy útiles para trabajar con cuerpos finitos, que serán necesarios para implementar el criptosistema. Se usará una versión online llamada CoCalc.

- Conexión a Internet: herramienta hardware necesaria para llevar a cabo el proyecto. Se necesita conexión a Internet para los recursos 2, 3 y 4.

3.3 Planificación inicial del proyecto

3.3.1 Estimación de horas por tarea

Se asignan una estimación de horas para cada tarea del proyecto para que el trabajador pueda organizarse y llegue al plazo previsto.

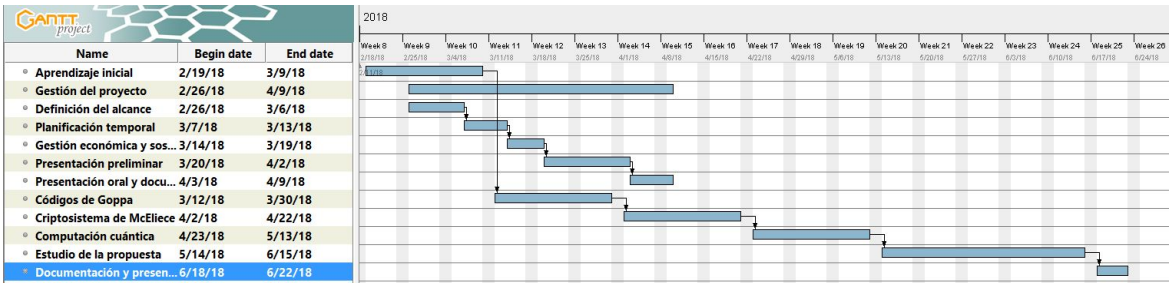
Tarea	Horas
Inicio del proyecto	**60**
1.Búsqueda	10
2.Aprendizaje inicial	50
Gestión del proyecto	**67**
1. Definición del alcance	25
2. Planificación temporal	8
3. Gestión económica y sostenible	10
4. Presentación preliminar	6
5. Presentación oral y documento final	18
Códigos de Goppa	**75**
1.Estudio y análisis	65
2.Ejemplos	10
Criptosistema de McEliece 1978	**75**
1.Estudio y análisis	65
2.Ejemplos	10
Computación cuántica	**75**

1.Estudio y análisis	75
Estudio de la propuesta	**125**
1.Estudio y análisis	95
2.Implementación	30
Documentación y presentación	**25**
1.Terminar la memória	10
2.Preparación de la defensa	15
Total	**427**

Tabla 1: Estimación de horas por tarea

3.3.2 Diagrama de Gantt

Este diagrama de Gantt engloba todas las tareas de la tabla anterior (sin contar los subapartados de los cuatro apartados principales):



Es debido mencionar que este es el diagrama de Gantt propuesto inicialmente. Se ha tenido que aplazar el proyecto; así que las nuevas fechas oscilan entre enero y abril (aproximadamente 4 meses para realizar el proyecto).

3.4 Identificación y estimación de los costes

El presupuesto del proyecto puede separarse en costes de recursos humanos, de hardware y de software. El presupuesto se calcula con la estimación de las tareas presentadas en la segunda entrega (teniendo en cuenta las horas estimadas).

3.4.1 Recursos humanos

El proyecto solo consta con un trabajador. El trabajador tendrá tres roles diferentes a lo largo del proyecto. A continuación se detallan las horas realizadas por cada rol con su precio por hora correspondiente.

Fase	Rol			Coste estimado
	Jefe de proyecto	Analista	Programador	
Planificación	77h	0h	0h	1.925€
Estudio general	0h	327h	0h	4.905€
Implementación del algoritmo	0h	0h	30h	450€
Total	**77h**	**327h**	**30h**	**7.280€**

Tabla 2: Horas y costes estimados de los cuatro roles para las 3 fases (la documentación entra en el trabajo del analista; ya que se realiza mientras se va desarrollando el trabajo)

	Rol		
	Jefe de proyecto	Analista	Programador
Coste por hora	25€/h	15€/h	15€/h

Tabla 3: Coste por hora de los roles

3.4.2 Hardware

También se necesita cierto hardware para llevar a cabo el proyecto. Son las herramientas que usará el trabajador para desarrollar el estudio, implementar el algoritmo y redactar la memoria. A continuación de muestra el hardware necesario con su coste correspondiente:

Producto	Precio	Unidades	Vida útil	Amortización/h	Horas	Amortización
Portatil Lenovo	1260€	1	3 años	0,21€/h	347h	72,87€
iPad 2	400€	1	3 años	0,067€/h	80h	5,36€

iPhone 6s	700€	1	3 años	0,117€/h	5h	0,59€
Total						**78,82€**

Tabla 4: Costes de hardware

Para calcular la amortización contamos con que un año tiene 250 días laborables con 8 horas al día laborables. El iPad 2 se usará para estudiar y leer documentación (una parte notable del proyecto, 80h). El iPhone 6s se usará principalmente para la comunicación del trabajador con el director del proyecto.

3.4.3 Software y licencias

El coste de la distribución de Microsoft Windows del portátil Lenovo va incluida en el precio del producto. Por lo tanto, su coste y amortización va incluido en la tabla anterior (tabla 3). El resto de software que se usa en el proyecto es gratuito (o se usará el plan gratuito) : Google Drive, CoCalc, webmail de la FIB, GanttProject.

3.4.4 Otros costes

Además de los costes calculados anteriormente, hay otros tres que se deben añadir al presupuesto total:

- Costes indirectos: En este apartado se debe considerar el coste de la oficina donde el trabajador desarrollará el proyecto. La oficina tiene cabidad para diez personas y cuesta 1000€ al mes. Por lo tanto, la parte que corresponde al trabajador es de 100€ al mes. Como el proyecto dura aproximadamente 4 meses, el precio total es de 400€ (el coste del consumo eléctrico, el agua y la conexión a internet va incluido en el precio).

- Contingencia: Este coste cubrirá cualquier posible variación en el plazo proyecto. Se ha fijado un nivel de contingencia del 10% (los riesgos del proyecto son muy leves y poco probables). Por lo tanto, tiene un coste de

- Imprevistos: No se espera ningún imprevisto en el proyecto. Todo los riesgos previstos que podrían surgir no conllevan ningún coste adicional.

3.4.5 Resumen de costes

RRHH	Hardware	Software	Indirectos	Contingencia	Imprevistos	**Total**
7.280€	78,82€	0€	400€	775,88€	0€	**8.534,7€**

Tabla 5: Resumen de costes

3.5 Control de gestión

El control de costes del proyecto se llevará a cabo en las reuniones periódicas con el director del proyecto. Se hará un informe de los costes después de cada fase (teniendo en cuenta el rol de cada fase y las horas empleadas) para llevar un control del presupuesto y que hayan variaciones.

Como ya se ha mencionado, no se prevén desviaciones ni imprevistos; el presupuesto del proyecto depende únicamente del tiempo de desarrollo (las horas en RRHH, el hardware y el coste de la oficina). Por lo tanto, si se cumple el plazo final de entrega, el presupuesto no tendrá ninguna desviación. En caso de extenderse el plazo, ya se cuenta con un 10% de margen de contingencia. Por otro lado, el proyecto difícilmente terminará antes de los previsto, ya que se pretende extender todo lo posible.

3.6 Sostenibilidad y compromiso social

	PPP	Vida útil
Ambiental	Consumo del diseño	Huella ecológica
	8/10	7/10
Económico	Factura	Plan de viabilidad
	9/10	9/10
Social	Impacto personal	Impacto social
	8/10	9/10
Rango de sostenibilidad	25/30	25/30

	50/60

Tabla 6:Matriz de sostenibilidad

3.6.1 Impacto ambiental

El proyecto no tiene demasiado impacto ambiental ya que es solo un estudio. Lo único a tener en cuenta es el consumo eléctrico de los dispositivos y los dispositivos en sí (si una vez terminada su vida útil se reciclan correctamente o reciben otro uso). Los tres dispositivos son de última generación y tienen un bajo consumo. Además, se reutilizan los recursos de hardware del trabajador. También se puede tener en cuenta el impacto ambiental del transporte (cuando el trabajador se desplaza a las reuniones); pero usará transporte público.

Realmente este proyecto no mejora ambientalmente nada respecto a otros ya existentes, ya que es un simple estudio y solo se necesita un ordenador.

3.6.2 Impacto económico

Los costes del proyecto son solo los recursos humanos (donde se destina la mayor parte del presupuesto), el hardware y la oficina del trabajador.

Como en el apartado anterior, no hay ninguna mejora respecto otros proyectos ya existentes; ya que los costes son inevitables (se necesita personal, hardware y un sitio de trabajo). Simplemente se reduce el presupuesto todo lo posible: personal relativamente barato, reutilización de hardware y una oficina de bajo coste.

3.6.3 Impacto social

A nivel personal, este proyecto me aporta darme cuenta de la importancia de la protección de la información y de lo rápido que avanza la tecnología. En un futuro próximo, cuando existan ordenadores cuánticos operativos, será imprescindible que se usen algoritmos de protección de datos más sofisticados y preparados para soportar ataques de ésta nueva tecnología.

Este problema aún no se ha resuelto. Por lo tanto, cualquier estudio que ayude a encontrar la mejor solución tiene un impacto a nivel mundial. Una mejora en la seguridad de la información es indirectamente una mejora social (calidad de vida). Por lo tanto, hay una necesidad real para el proyecto; ya que contribuye en una de las propuestas para encontrar una solución a este problema.

4. CONCLUSIONES

El tema principal de este proyecto ha sido estudiar los códigos binarios de Goppa y el criptosistema de McEliece, con el objetivo de entender su funcionamiento y su alto grado de seguridad. A pesar de que el criptosistema tiene más de 40 años y apenas recibe uso, se ha presentado un criptosistema basado en el de McEliece en el concurso del NIST ya mencionado. Por lo tanto, hemos realizado un apartado sobre esta propuesta, ya que puede llegar a tener un futuro en la computación cuántica.

Para realizar la parte práctica del trabajo (los ejemplos y la implementación) ha sido de gran ayuda la herramienta mencionada en el apartado *3.2.2* llamada CoCalc, ya que tiene librerías sobre cuerpos finitos. Se ha usado a modo de calculadora para hacer los ejemplos a mano, y para hacer una implementación casi completa. Usamos el término "casi" porque hemos tenido una dificultad a la hora de usar incógnitas de elementos del cuerpo finito; en concreto, las *sigmas* del polinomio localizador de errores (paso 2 del algoritmo para corregir errores, apartado *2.1.7* y *2.1.8*). Sin embargo, hay una implementación completa del algoritmo aparentemente funcional en [17]. Se puede encontrar nuestra implementación en el apéndice al final del proyecto.

Inicialmente, se había planteado estudiar y hacer un apartado sobre la computación cuántica; ya que pensábamos que iba a ser necesario para entender la propuesta. Pero después de leerla, vimos que en ningún momento era necesario ningún conocimiento sobre ello para entender su contenido. Por esta razón, decidimos no incluir este apartado en el proyecto.

Para concluir, queremos hacer énfasis en el gran potencial que tiene este criptosistema. Se ha estudiado su seguridad durante muchos años y siempre ha demostrado un alto nivel de seguridad. Y en cuanto a la computación cuántica, parece no haber ningún algoritmo conocido que pueda romper este criptosistema con facilidad. Así que a pesar de tener ciertos puntos débiles en cuanto a la eficiencia, quizás tenga un futuro en el mundo de la seguridad informática.

BIBLIOGRAFÍA

[1] McEliece, R. J. «A public-key cryptosystem based on algebraic coding theory» DSN Progress Report 42, 114-116, 1978.

[2] Sanyols, N.; Xambó, S. «Codis correctors d'errors i criptografia postquàntica». Boletín de la Sociedad Catalana de Matemáticas. Vol. 33, núm. 2, 147-171, 2018.

[3] Jochemsz, E. «Goppa Codes & the McEliece Cryptosystem». Tesis para la obtención de un doctorado en matemáticas, en la Facultad de Ciencias, División de Matemáticas e Informática de la Universidad VU de Ámsterdam, 2002.

[4] Shor, P. W. «Algorithms for quantum computation: discrete logarithms and factoring». A: 35th Annual Symposium on Foundations of Computer Science. Los Alamitos, CA: IEEE Comput. Soc. Press, 124-134, 1994.

[5] Sendrier, N. «Cryptosystèmes à clé publique basés sur les codes correcteurs d'erreurs». Mémoire d'habilitation à diriger des recherches. París: Université Pierre et Marie Curie, Paris 6; Rocquencourt: Institut National de Recherche en Informatique et Automatique, 2002.

[6] Leon, J. S. «A probabilistic algorithm for computing minimum weights of large error-correcting codes». IEEE Transactions on Information Theory, 34(5), 1354-1359, 1988.

[7] Lee, P. J.; Brickell, E. F. «An observation on the security of McEliece's public-key cryptosystem». In Christoph G. Günther, editor, Advances in cryptology - EUROCRYPT '88, volume 330 of Lecture Notes in Computer Science, 275-280, 1988.

[8] Stern, J. A method for finding codewords of small weight. In Gerard D. Cohen and Jacques Wolfmann, editors, Coding theory and applications, volume 388 of Lecture Notes in Computer Science, 106-113, 1989.

[9] van Tilburg, J. «On the McEliece public-key cryptosystem». In Shafi Goldwasser, editor, Advances in cryptology - CRYPTO '88, volume 403 of Lecture Notes in Computer Science, 119-131, 1990.

[10] Canteaut, A.; Chabanne, H. «A further improvement of the work factor in an attempt at breaking McEliece's cryptosystem». In P. Charpin, editor, EUROCODE 94, 1994.

[11] Canteaut, A.; Chabaud, F. «A new algorithm for finding minimumweight words in a linear code: application to McEliece's cryptosystem and to narrow-sense BCH codes of length 511». IEEE Transactions on Information Theory, 44(1), 367-378, 1998.

[12] Canteaut, A.; Sendrier, N. «Cryptanalysis of the original McEliece cryptosystem». In Kazuo Ohta and Dingyi Pei, editors, Advances in cryptology - ASIACRYPT '98, volume 1514 of Lecture Notes in Computer Science, 187-199, 1998.

[13] Bernstein, D. J.; Chou, T.; Lange, T.; von Maurich, I.; Misoczki, R.; Niederhagen, R.; Persichetti, E.; Peters, C.; Schwabe, P.; Sendrier, N.; Szefer, J.; Wang, W. «Classic McEliece: conservative code-based cryptography», noviembre de 2017.

[14] Bernstein, D. J. «Grover vs McEliece». En Sendrier [15], 73-80.

[15] Sendrier, N. (editor). «Post-Quantum Cryptography», Third International Workshop, PQCrypto 2010, Darmstadt, Germany, May 25-28, 2010. Proceedings, volume 6061 of Lecture Notes in Computer Science. Springer, 2010.

[16] Bernstein, D. J.; Lange, T.; Peters, C. «Attacking and defending the McEliece cryptosystem». In Johannes A. Buchmann and Jintai Ding, editors, Post- Quantum Cryptography, Second International Workshop, PQCrypto 2008, Cincinnati, OH, USA, October 17-19, 2008, Proceedings, volume 5299 of Lecture Notes in Computer Science, 31-46, 2008.

[17] Risse, T. «How SAGE helps to implement Goppa Codes and McEliece PKCSs». DSI GmbH Bremen as well as Institute of Informatics & Automation (IIA), Faculty EEE & CS, Hochschule Bremen, University of Applied Sciences, 2011.

[18] Engelbert, D.; Overbeck, R.; Schmidt, A. «A summary of McEliece-type cryptosystems and their security». TU-Darmstadt, Department of Computer Science, Cryptography and Computer Algebra Group, 2006.

[19] Repka, M.; Zajac, P. «Overview of the McEliece cryptosystem and its security». Tatra Mountains, Mathematical Publications, 57-83, 2014.

[20] Dinh, H.; Moore, C.; «Russell, A. McEliece and Niederreiter Cryptosystems That Resist Quantum Fourier Sampling Attacks». En orden de los autores: Indiana University South Bend, University of New Mexico, and Santa Fe Institute, University of Connecticut. 2011.

[21] Calculadora de kernel de matrizes. URL: https://www.mathdetail.com/null.php

[22] Miranda, N. D. «Computación cuántica». Universidad de La Laguna, 2015.

[23] Computer Security Division, Information Technology Laboratory, National Institute of Standards and Technology, and Department of Commerce. "Post-Quantum Cryptography | CSRC." Computer Security Division, Information Technology Laboratory, National Institute of Standards and Technology, U.S. Department of Commerce. URL: https://csrc.nist.gov/Projects/Post-Quantum-Cryptography

[24] Omicrono. "El Nuevo Procesador Cuántico De Google Puede Solucionar El Gran Problema De Esta Tecnología". 05-3-2018. URL: https://omicrono.elespanol.com/2018/03/bristlecone-procesador-cuantico-de-google/

[25] Wikipedia. "National Institute of Standards and Technology". 10-4-2018. URL: https://en.wikipedia.org/wiki/National_Institute_of_Standards_and_Technology

[26] SAFEcrypto. "Post-Quantum Crypto Lounge - SAFEcrypto". URL: https://www.safecrypto.eu/pqclounge/

[27] Both, L.; May, A. «Decoding Linear Codes with High Error Rate and its Impact for LPN Security». Horst G¨ortz Institute for IT-Security, Ruhr-University Bochum, Germany, Faculty of Mathematics, 2018.

APÉNDICE

Implementación

1. Algoritmo de corrección

1.1 Montar código de Goppa y las matrices

```
import sympy
from sympy import Matrix

def inv(i, pg):
  aa = xgcd(z-a^i,pg)[1]
  return aa

def codificar(msg): #msg tiene que ser una matriz 1xk
  cwa = Matrix(msg)*Matrix(mg)
  cw = []
  for i in cwa:
    cw.append(i%2)
  return cw

def anadir_errores(e, cw):
  for e in errores:
    cw[e-1] = (cw[e-1] + 1)%2
  return cw

def calcular_sindrome(cwe):
  s = 0
  for i in range(len(cwe)):
    if cwe[i] == 1:
      s = s + h2[i]
  return s

z = PolynomialRing(GF(2),'z').gen()

m = 4
n = 12
```

```
mod = z^4+z+1

F16 = GF(2^m, 'a', modulus=mod)
a = F16.gen()

g = (z+a^13)*(z+a^14) #polinomio generador

h = [] #matriz de paridad en forma de array
for i in range(1,n+1):
  h.append(inv(i,g))

hc = [] #matriz de paridad separada por coeficientes
for i in range(len(h)): #forma hc
  hc.append(h[i].coefficients(sparse=false))

f, c = len(hc[0])*4, n #filas y columnas de la matriz de paridad binaria
hb = [[0 for x in range(c)] for y in range(f)] #matriz de paridad en forma binaria
for x in range(c): #forma hb
  for p in range(int(f/4)):
    for y in range(len(hc[x][p].polynomial().list())):
      hb[y+(p*4)][x] = int(hc[x][p].polynomial().list()[y])

mga = Matrix(hb)
mga = mga.nullspace()
mg = [[(mga[y][x])%2 for x in range(12)] for y in range(4)] #matriz generadora
en forma binaria
print "Dimensión k del código: " + str(len(mg)) #te muestra la dimensión del
código por pantalla

g2 = g^2 #polinomio generador al cuadrado
h2 = [] #matriz de paridad de g2(útil para hacer operaciones como buscar el
síndrome)
for i in range(1,n+1):
  h2.append(inv(i,g2))
```

Las 4 líneas en negrita es la información necesaria que el usuario debe añadir para montar el código de Goppa (n, m, mod, y g).

1.2 Codificar el mensaje y calcular el síndrome

```
msg = [[1,1,0,0]] #añades el mensaje a codificar

cw = codificar(msg)
print "Palabra del código: " +  str(cw) #te muestra la palabra del código por pantalla
print "Añade un total de " + str(len(hc[0])) + " errores." #en forma de array

errores = [4,5] #añades los errores

cwe = anadir_errores(e, cw) #palabra del código con los errores
print "Palabra del código con errores: " + str(cwe)
s = calcular_sindrome(cwe)
print "Síndrome de la palabra: " + str(s)
```

Las 2 líneas en negrita es la información que el usuario añade. El mensaje a cifrar y los errores a añadir.

Como se puede apreciar en el código, tiene los valores del ejemplo *2.1.8*. Y este es el output correspondiente a este ejemplo:

```
Dimensión k del código: 4
Palabra del código: [0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0]
Añade un total de 2 errores.
Palabra del código con errores: [0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0]
Síndrome de la palabra: (a^3 + a^2 + 1)*z^3 + (a^2 + 1)*z^2 + (a^3 + a^2 + 1)*z
```

Para saber la potencia de α se puede mirar la tabla de potencias o ejecutar, por ejemplo:

```
Input: (a^3 + a^2 + 1).log(a)
Output: 13
```

Una vez obtenido el síndrome, solo es necesario multiplicarlo por el polinomio localizador de errores y resolver el sistema de ecuaciones. Hemos tenido un problema al operar con incógnitas del cuerpo finito (las sigmas del polinomio localizador de errores). Sin embargo, hay una implementación de otro algoritmo, aparentemente funcional, que corrige los errores en [17].

2. Criptosistema de McEliece

La parte importante de este código es generar las matrices aleatorias S invertible y P de permutación.

Generar la matriz de permutación es bastante sencillo. Existe la función en diferentes librerías. Por ejemplo, en MatLab se genera de tal forma:

```
A = eye( N )  # es mejor usar A = speye( N ) cuando N es muy grande
idx = randperm(1:N)
A = A(idx, :)
```

Generar la matriz invertible S es algo más complicado. En [2] hay un algoritmo que genera matrices invertible aleatoria uniformemente distribuida.