Escuela
Politécnica
Superior

Sistema seguro para la gestión de tareas en proyectos

Grado en Ingeniería Informática

Trabajo Fin de Grado

Autor:
Adrián Caparrós Carrillo
Tutor/es:
Jose Vicente Aguirre Pastor

Universitat d'Alacant
Universidad de Alicante

Mayo 2022

Sistema seguro para la gestión de tareas en proyectos

Autor
Adrián Caparrós Carrillo

Tutor/es
Jose Vicente Aguirre Pastor
Ciencia de la computación e Inteligencia Artificial

Grado en Ingeniería Informática

Escuela
Politécnica
Superior

Universitat d'Alacant
Universidad de Alicante

ALICANTE, Mayo 2022

Agradecimientos

Me gustaría comenzar agradeciéndole a mi tutor todo el apoyo que me ha ofrecido en todo momento durante el desarrollo del proyecto. También, por ser quién me hizo descubrir el mundo de la criptografía en sus clases y haberme ampliado mis conocimientos en el transcurso de este proyecto. Por enseñarme no solo algoritmos, sino también a ser mejor ingeniero.

A mis padres, porque sin ellos en ningún momento habría podido estar escribiendo estas palabras. Porque pese a los malos momentos, no solo no me dejaron abandonar este sueño, sino que siempre estuvieron ahí apoyándome y dándomelo todo. No puedo sentirme más afortunado de teneros, ojalá que este sea el primer paso para devolveros todos los sacrificios que habéis hecho por mí.

Y por último, agradecértelo a ti María, eres el motor de mi vida y siempre consigues transmitirme tu energía. Desde que has estado a mi lado, has sido un modelo a seguir para mí y sin ti no habría llegado hasta donde estoy. Verte esforzarte para lograr tus sueños, me hace sentir que yo también puedo lograr lo que me proponga. Muchísimas gracias por no haberte rendido nunca ni tirado la toalla con un desastre como yo.

Este trabajo se lo dedico a mis abuelos,
porque vosotros siempre estuvisteis en este sueño que estoy a punto de cumplir.

Era su lema.
Parte de su lema, al menos.
"Vida antes que muerte. Fuerza antes que debilidad. Viaje antes que destino".

Brandon Sanderson, El Camino de los Reyes

Índice general

Índice de figuras

Índice de tablas

Índice de Códigos

1 Introducción

Ha comenzado a resultar algo común, ver día a día como los medios de comunicación nos alertan de la sucesión de multitud de delitos o amenazas informáticas, siendo estos tan solo los más notados. La capacidad de ataque ha ido creciendo de manera exponencial, dando lugar a situaciones cada vez más complejas y costosas de defender, lo que hoy en día comienza a conocerse comúnmente como ciberamenazas o ciberataques, ha dejado de ser algo simplemente utilizado en la ficción para convertirse en una realidad a la que cualquier persona conectada a internet está expuesta.

Estos hechos quedan reflejados en los datos e informes que proporcionan organismos e instituciones como el CCN-CERT (Centro Criptológico Nacional Computer Emergency Response Team CCN-CERT, 2021) o el INCIBE (Instituto Nacional de Ciberseguridad de España INCIBE, 2022).

La perspectiva acerca del valor de la información con la que se trabaja a diario en cualquier sistema informático, que se puede considerar que no es relevante porque se tiende a creer que nadie más tendrá acceso a esta, cambia radicalmente si la misma cae en manos equivocadas, las fugas de información se dan en todos los niveles desde grandes empresas hasta usuarios particulares.

El problema de estas fugas se da cuando la información sustraída se encontraba almacenada sin ningún tipo de protección, es por ello por lo que en cualquier sistema informático hoy en día es vital garantizar la confidencialidad de la información con la que se trabaja en el mismo.

Pero proporcionar seguridad en un sistema a los usuarios no debe limitarse tan solo a dotar de confidencialidad a la información con la que trabajen, también se deben considerar otros aspectos como el poder garantizarles la autenticidad de la información, gracias a la cual los usuarios del sistema pueden confiar en la procedencia u origen de la información que visualizan.

Por otra parte, también resulta vital asegurar que la información no ha sido alterada, es decir, garantizar la integridad de la información. Todas estas necesidades conllevaron el surgimiento de una ciencia conocida como criptografía.

Desde tiempos remotos, se han empleado mensajes secretos para ocultar información a personas no autorizadas a esta, la criptografía desde la clásica hasta la moderna ha acompañado a los seres humanos, amoldándose a las exigencias y requisitos de los nuevos tiempos, donde cada vez las amenazas son más y más complejas de solventar, pese a ello la criptografía sigue logrando darnos una capa de seguridad para protegernos.

Este hecho queda reflejado en la afirmación que hicieron Shafi Goldwasser, Silvio Micali, Adi Shamir y Ronald Rivest: "la criptografía es una de las ramas más pesimistas de la ciencia: asume la existencia de adversarios extremadamente poderosos que pueden leer todos tus mensajes, generar mensajes falsos o modificar tus contraseñas y tus bits aleatorios. Sin embargo, también es una de las ramas más optimistas de la ciencia, ya que muestra cómo puedes superar esas dificultades con el poder de las matemáticas y de algoritmos computacionales como los que nosotros y nuestros colegas hemos desarrollado" (González Salomone, 2018-06-13).

1.1 Motivación

La principal fuente de motivación para la realización de este trabajo surge de la propia naturaleza de la criptografía y en como está de manera imperceptible para los usuarios, es capaz de lograr hacer que una información sea perfectamente visible para las personas autorizadas a esta y al mismo tiempo ocultársela a aquellos que no deben acceder a la misma.

Mi primera toma de contacto con la criptografía se dio con motivo de cursar la asignatura Estrategias de seguridad del itinerario Tecnologías de la información del grado Ingeniería Informática, el cual estoy cursando, despertando un interés que ha promovido el comenzar a interesarme por un aspecto hasta entonces prácticamente desconocido.

Aunque cada día empieza a sonar más la importancia de la seguridad en el mundo digital, para muchos aún sigue siendo un mundo por conocer. El mundo de la ciberseguridad en general se ha convertido en una fascinación personal, un mundo donde la criptografía es una parte fundamental de este.

A todo esto se le suma mi experiencia en el periodo que realice prácticas externas, donde pude apreciar mejor el uso que se hacía de un gestor de tareas en un entorno compartido y observe comportamientos o funcionalidades a las que se les podría dotar de mayor seguridad mediante técnicas criptográficas.

Es por todo lo mencionado anteriormente, acerca de la importancia que supone contar con un sistema con propiedades como la confidencialidad, autenticidad e integridad de la información, por lo que se optó por desarrollar un software cliente/servidor. Con el objetivo puesto en permitir a cualquier usuario de este, hacer una gestión de las tareas en cualquier ámbito, de manera segura. Para ello, se ha efectuado un análisis e implementación de las técnicas y algoritmos criptográficos que permiten dotar de seguridad a un sistema informático en todos sus aspectos.

2 Marco Teórico

En este capítulo se detallarán a nivel teórico las diferentes técnicas y algoritmos criptográficos que se han aplicado en el proyecto. Además, se comentará el proceso de gestión de tareas y las herramientas más destacables para llevarla a cabo.

2.1 Sistemas Seguros

Cuando se menciona que un sistema es seguro, se busca destacar que dicho sistema es infalible ante cualquier clase de peligro o amenaza, y que entre muchas cosas es capaz de garantizar la protección de la información.

En la práctica resulta imposible o muy difícil lograr un sistema realmente seguro, por ello en lugar de eso se recurre al término de la fiabilidad de un sistema.

Para poder garantizar dicha fiabilidad, un sistema debe apoyarse sobre los tres principios o pilares básicos de la seguridad de la información, la confidencialidad, la disponibilidad y la integridad de la información.

- La confidencialidad de la información es la cualidad por la cual la información de un sistema no debe ser accesible a las personas o sistemas que no tengan autorización para acceder a esta.
- La integridad de la información se fundamenta en garantizar que la información no ha sido alterada por ningún usuario que no esté autorizado a dicha información.
- Por último, la disponibilidad de la información busca garantizar que los usuarios o elementos autorizados, tengan disponible en cualquier momento la información que solicitan.

A su vez, dado que hoy en día vivimos en un mundo interconectado, cualquiera puede afirmar ser el dueño u origen de la información que recibimos. La propiedad conocida como autenticidad de la información, busca garantizar a los usuarios conocer en todo momento con certeza el origen o propietario de un elemento u acción.

Esta propiedad se encuentra muy ligada con otra conocida como No Repudio, con la cual se evita que un usuario pueda negar la autoría de una información o acción. Dicha propiedad puede verse de dos formas:

- No repudio en origen: Cuando el emisor es incapaz de negar que realizo un envío de información, dado que el receptor dispone de pruebas inapelables.

- No repudio en destino: En este caso, el emisor dispone de pruebas irrefutables de la recepción de la información que envió al receptor.

Las amenazas informáticas suelen englobarse bajo tres grupos principales en función del carácter de la amenaza, desde amenazas lógicas, pasando por catástrofes hasta personas. Para paliar estas amenazas surgen un conjunto de mecanismos repartidos en tres grupos, mecanismos de detección, de recuperación y de prevención. En este último grupo, es donde entra la criptografía.

Con técnicas como el cifrado de la información, se consigue que el coste de realizar un ataque a un sistema sea más elevado que el coste de lo que se puede lograr obtener de este. Además de permitir garantizar la confidencialidad de la información, también permite dotar a un sistema informático de las propiedades comentadas anteriormente.

2.2 ¿Qué es la criptografía?

La criptografía no es algo concreto del ámbito digital, lleva con la humanidad desde tiempos remotos, forma parte de una ciencia antigua conocida como criptología, cuya definición según la RAE es la siguiente: "Estudio de los sistemas, claves y lenguajes ocultos o secretos." (RAE, s.f.-b), es decir, es la disciplina que se encarga de los problemas teóricos relacionados con la seguridad en el intercambio de mensajes entre un emisor y un receptor. Dicha ciencia está dividida a su vez en dos ramas, cada una de las cuales con un enfoque totalmente opuesto.

Por un lado, se encuentra la ya mencionada criptografía, encargada del estudio de las técnicas que se utilizan para lograr proteger la información y las comunicaciones, la RAE la define como "Arte de escribir con clave secreta o de un modo enigmático." (RAE, s.f.-a).

Por el otro lado se encuentra la criptología, que es la encargada del estudio sobre como romper sistemas criptográficos.

En la criptografía, cuando se juntan un conjunto de algoritmos criptográficos para proporcionar una funcionalidad referente a la seguridad, se les denominan criptosistemas.

Un criptosistema, permite que al aplicar una serie de funciones de cifrado y descifrado, se puedan realizar envíos de información por canales inseguros. Se debe considerar como medio inseguro no solo un medio de transmisión, sino también el propio almacenamiento final de la información.

Todo criptosistema consta de cinco componentes:

- M: Es el conjunto de todos los mensajes en claro (denominación que reciben los mensajes previamente a ser cifrados). Por lo que se tiene que todo texto en claro $m \in M$.
- C: Es el conjunto de todos los criptogramas (reciben esta denominación los mensajes tras haberles aplicado una transformación de cifrado). Dándose que todo criptograma $c \in C$.

- K: Es la representación del espacio o conjunto de claves, aplicables al criptosistema.

- E: Conjunto de todas las transformaciones de cifrado definidas por el algoritmo dado, que son comunes y que se aplican a cada m, dando como resultado c. Existe una transformación E_k diferente para cada clave $k \in K$. Con esto se deduce la siguiente expresión de cifrado: $E_k(m) = c$.

- D: Es el conjunto de las transformaciones de descifrado, al contrario que en el anterior paso, en este caso se aplican sobre un c, devolviendo el m original. Existe igualmente una transformación D_k diferente para cada clave $k \in K$. Y en este caso la expresión de descifrado resultante sería: $D_k(c) = m$.

Dadas las dos expresiones mencionadas anteriormente, se debe cumplir la siguiente condición:

$$D_k(E_k(m)) = m$$

De forma visual se pueden ver los diferentes elementos que componen un criptosistema en la figura 2.1. Se puede observar fácilmente como la k debe ser enviada siempre por un canal seguro, dado que debe permanecer siempre oculta de posibles atacantes, en cambio, el criptograma si puede ser enviado por un medio inseguro.

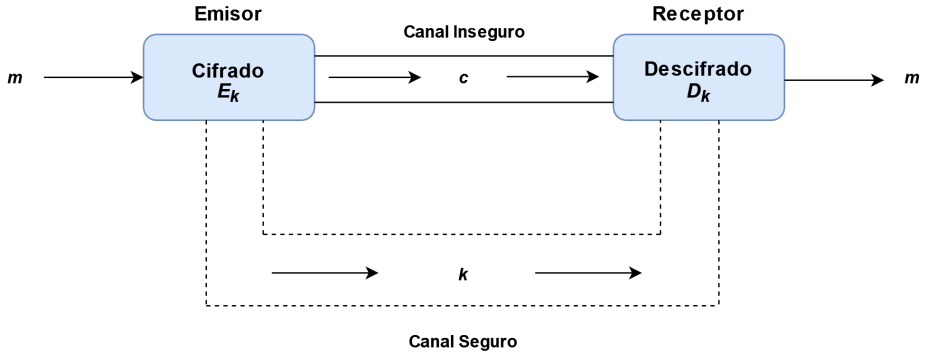

Figura 2.1: Esquema elementos criptosistema.

De los cinco componentes que se han mencionado, el elemento más importante en todo criptosistema siempre será la clave. En lo referente a esta, en todo sistema se debe garantizar que el número de claves conocidas como débiles en el mismo sea nulo o muy pequeño respecto al espacio total de claves.

El término de clave débil se usa para hacer referencia a claves $k \in K$, que al utilizarlas, podrían dar lugar a dos situaciones que provocarían una brecha en la seguridad del criptosistema, dichas situaciones son las siguientes:

$$E_k(M) = M \qquad\qquad E_k(E_k(M) = M$$

Evitar el uso de este tipo de claves es de gran importancia, puesto que para todo criptosistema, la clave es el elemento fundamental que proteger. Dado que es en esta, donde reside toda la seguridad de un criptosistema. Esto se refleja en la afirmación del lingüista y criptógrafo holandés Auguste Kerckhoffs en su libro La Cryptographie Militaire, "La seguridad de un criptosistema no debe depender de mantener secreto el algoritmo de cifrado (seguridad por oscuridad). La seguridad solo debe depender de mantener una pequeña información secreta, la denominada clave de cifrado" (Auguste, 1883).

Toda la teoría sobre la que se fundamentan los criptosistemas recae sobre tres pilares, la teoría de los números, la teoría de la complejidad y por último la teoría de la información. Esta última se sustenta sobre los estudios realizados por Claude Shannon (para más información, véase Shannon, 1949). En los cuales se destacan dos técnicas que se utilizan para ocultar la redundancia en un texto en claro. Las técnicas son las siguientes:

- Difusión: Sirve para evitar que aparezcan propiedades estadísticas del lenguaje sobre el c resultante de cifrar un m. Se apoya en técnicas de transposición.
- Confusión: Busca encubrir la vinculación entre m y el c obtenido del proceso de cifrado correspondiente. Hace uso de técnicas de sustitución.

Dentro de los criptosistemas según distintos criterios se pueden llevar a cabo las siguientes distintas clasificaciones, la más simple de todas es según la antigüedad, donde se dividen en clásicos y modernos. El punto de inflexión de uno a otro es la aparición de la informática. Otra clasificación se da en función de la manera de cifrar la información, donde se tienen dos tipos:

- Cifradores en flujo: Se cifra bit a bit o carácter a carácter.
- Cifradores en bloque: La información se agrupa en un conjunto de bits, caracteres, etc.

Para ver más detalle sobre estos dos tipos, véase los capítulos 10 y 11 del libro de Lucena (Lucena López, 2015). La clasificación más conocida se da en función de la clave utilizada para cifrar y descifrar, dividiéndose nuevamente en dos tipos:

- Cifradores asimétricos: Utilizan una única clave para el proceso de cifrado y descifrado.
- Cifradores simétricos: Emplean dos claves, una para el cifrado y otra para el descifrado.

2.3 Cifrado Simétrico

Los algoritmos de cifrado simétrico, como se ha comentado anteriormente, son aquellos que emplean una misma clave k tanto para el cifrado como para el descifrado de la información. Esto deriva en un grave inconveniente al ser empleados en comunicaciones, dado que supone enviar desde el emisor hacia el receptor la clave utilizada por un canal seguro. Este uso de la misma clave en ambos procesos se puede ver de manera visual en la figura 2.2.

Dentro de la familia de los algoritmos simétricos, se distinguen entre los que son cifradores en flujo y los que son cifradores en bloque. Estos últimos son los que se apoyan en las técnicas

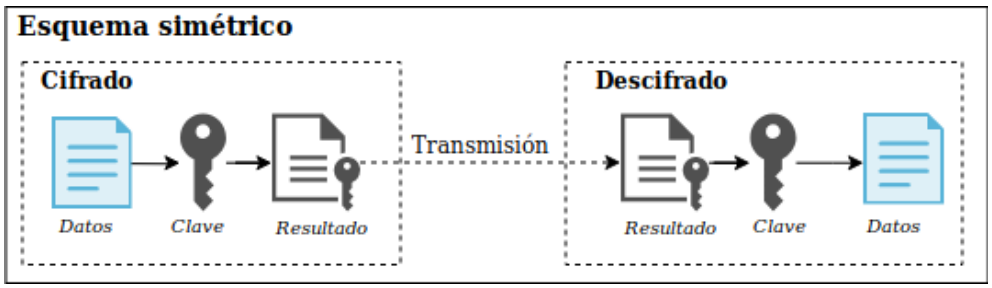

Figura 2.2: Esquema cifrado simétrico (*Cifrado: ¿simétrico o asimétrico?*, s.f.).

de confusión y difusión propuestas por Shannon. Uno de los más destacados algoritmos de este tipo es Rijndael, que hoy en día se le conoce más comúnmente, como el estándar avanzado de cifrado (AES).

2.3.1 AES

Rijndael es un algoritmo simétrico de cifrado por bloques. Fue diseñado para permitir tanto bloques como claves con un tamaño variable siempre que dicho tamaño sea múltiplo de 4 bytes, aunque el tamaño mínimo recomendable es de 128 bits en ambos casos. El algoritmo opera a nivel de byte, realizando una interpretación de estos como si fueran elementos de un cuerpo de Galois $GF(2^8)$ (para mas información, véase *Grupo de Galois*, s.f.).

El algoritmo Rijndael fue el ganador de un proceso de selección organizado por el $NIST$ (Instituto nacional de estándares y tecnología). En el proceso participaron distintos algoritmos con el fin de convertirse en el nuevo estándar avanzado de cifrado. Dicho proceso destacó particularmente por el hecho de ser un concurso abierto a todo el mundo. Además, contó con un proceso de selección totalmente transparente y hubo una participación total de toda la comunidad criptográfica en su análisis.

El ganador termino siendo un algoritmo que disponía de la confianza de todos para convertirse en AES. La mayor prueba de esto es que, además, de las recomendaciones de utilizarlo por parte del $NIST$, incluso la misma NSA (Agencia de seguridad nacional), aprueba su utilización en el cifrado de información clasificada en el estándar $FIPS\ PUB$ 197 (M. Dworkin y cols., 2001-11-26).

Al ser elegido vencedor se estableció, para AES, una longitud de bloque fija de 128 bits. En lo referente a la clave se estipularon tres longitudes variables donde tenemos los siguientes tamaños: 128, 192 y 256 bits.

2.3.2 Estructura de AES

En lo referente a la estructura del algoritmo, la base de este es la sucesión de una serie de rondas en las cuales se aplican un conjunto de 4 funciones matemáticas diferentes e invertibles. Al contrario que su predecesor DES, el cual era un cifrador tipo Feistel (denominación

que hace referencia a los cifradores en bloque, en los que el bloque se divide en dos mitades y en cada una de las rondas las funciones se aplican solo sobre una única mitad del bloque y en la siguiente sobre la mitad restante).

AES, sin embargo, es de tipo no Feistel por lo que en este el proceso de cifrado se realiza sobre el bloque completo en cada ronda. Además, dado el diseño de la estructura del algoritmo, sus creadores afirman que es muy improbable encontrar claves débiles en este.

AES utiliza, tanto para el bloque de información como para la clave, una representación en forma de matrices rectangulares. La matriz que contiene el bloque de información es conocida como matriz de estado y se le van aplicando las diferentes funciones que componen AES en cada una de las rondas. Dicha matriz posee 4 filas y N_b columnas, siendo N_b = tamaño del bloque en bits / 32. En cuanto a la matriz que contiene la clave, al igual que la anterior, posee 4 filas y N_k columnas, donde N_k = tamaño de la clave en bits / 32.

En Rijndael el número de rondas depende tanto del tamaño del bloque como del tamaño de la clave, en el caso de AES al tener el tamaño de bloque fijo en 128 bits, las rondas solo dependen de la clave como se muestra en la tabla 2.1

Tamaño de la clave	Número de rondas
AES-128	10
AES-192	12
AES-256	14

Tabla 2.1: Número de rondas en AES para distintos tamaños de clave

En cada una de las rondas se utiliza una función denominada expansión de clave. Con la cual partiendo de la clave inicial se consiguen generar distintas subclaves que se aplicaran en cada una de las rondas. En estas rondas se llevará a cabo la aplicación de las siguientes funciones sobre la matriz de estado:

- *SubBytes*: Se realiza una sustitución de los 16 bytes que representan a la matriz de estado mediante una tabla conocida como S-box.
- $ShiftRows$: Consiste en permutar las filas de la matriz de estado de la siguiente forma, la primera fila no rota ningún byte, la segunda rota 1, la tercera 2 y la cuarta rota 3.
- $MixColumns$: Se efectúan multiplicaciones de cada una de las columnas de la matriz de estado por un polinomio fijo.
- $AddRoundKey$: Realiza la suma XOR (para mas información, véase *Operador a nivel de bits*, s.f.) de la matriz de estado con los valores de la matriz de la subclave correspondiente a la vuelta.

Todas estas funciones son invertibles. Por ello, para el descifrado, se aplican las inversas de esas funciones utilizando el mismo número de rondas. Con respecto a las subclaves, se

emplean las mismas, pero esta vez empezando por la última primero.

De manera general puede verse el proceso de cifrado en la figura 2.3. En la cual se puede ver cómo y cuándo se aplican las diferentes funciones en cada una de las rondas. También se puede ver el proceso de generación de las subclaves usadas en cada ronda. Por último, destacar que existe una ronda inicial en la cual solo se aplica la función *AddRoundKey* y que en la última de las rondas no se aplica la función *MixColumns*.

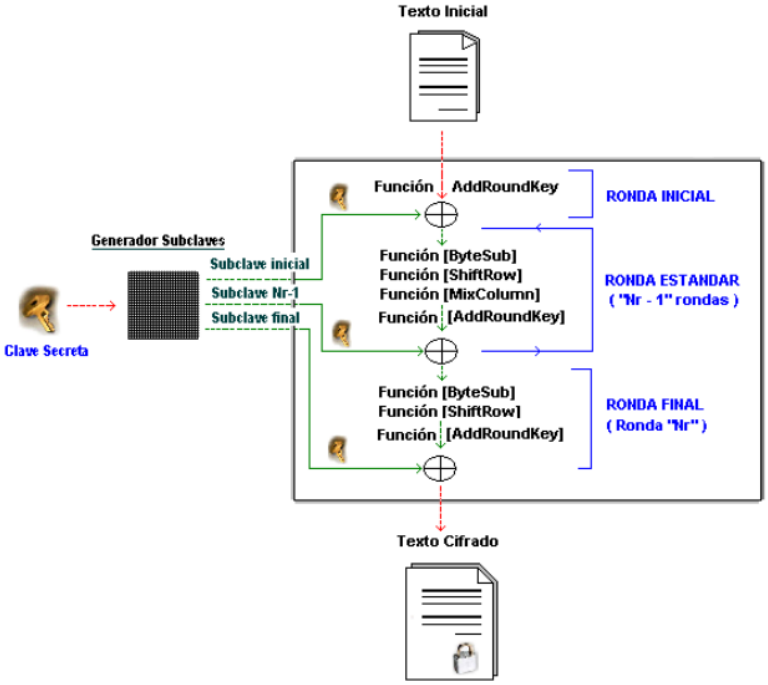

Figura 2.3: Esquema cifrado AES (Muñoz, 2004).

2.3.3 Modos de operación y padding en cifradores en bloque

Para poder aplicar el cifrado sobre un bloque de información en cualquier algoritmo de tipo cifrador en bloque, es necesario que dicho bloque sea del tamaño establecido por el algoritmo. En AES el tamaño está fijo a 128 bits como se ha comentado anteriormente, es por ello que en ocasiones se hará necesario el uso de lo que se conoce como padding o relleno.

Existen varios métodos de aplicar dicho relleno. En concreto, en AES la tendencia suele llevar a incluir como relleno un valor hexadecimal que indique el número de bytes que faltan para completar el tamaño de bloque esperado. En AES, dicho relleno siempre existirá, incluso cuando se disponga de un bloque de 128 bits. Para este último caso se cifrarán dos bloques, donde el segundo será el relleno.

Otro aspecto relevante en el cifrado de algoritmos simétricos con cantidades de información superiores al tamaño del bloque es la necesidad de la utilización de lo que se conoce como modos de cifrado. Estos son básicamente descripciones de cómo aplicar de manera sucesiva las operaciones de cifrado pertinentes sobre grandes cantidades de información. Existen diversos modos de cifrado, donde el modo básico es el denominado ECB.

El modo de cifrado ECB (*Electronic CodeBook*), es el más fácil de utilizar, puesto que simplemente realiza un cifrado de cada uno de los bloques con la clave k de manera separada cada uno, como se puede ver en la figura 2.4. Este modo de funcionamiento conlleva a que en la práctica su uso nunca sea recomendado. Dado que las desventajas que presenta son enormes, algunas de las desventajas más notables son:

- Si se cifran bloques con la misma información, el bloque resultante siempre será el mismo, esto trae consigo lo que se denominan como ataques de diccionario (para mas información sobre este tipo de ataques, véase *Ataque de diccionario*, s.f.).
- Un atacante en un canal inseguro, podría capturar los bloques y efectuar modificaciones en los bloques deseados.

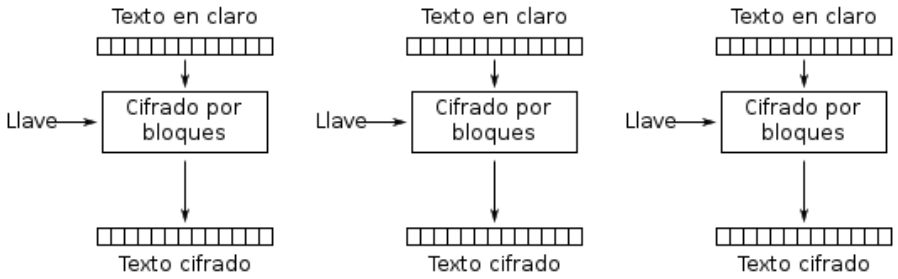

Figura 2.4: Modo de operación ECB (*Block cipher mode of operation*, s.f.).

El gran problema como se puede observar reside en que no existe una retroalimentación en el proceso de cifrado de los bloques. Por este motivo, se han diseñado otros modos de cifrado en los que el cifrado de bloques anteriores, si condiciona el cifrado del bloque actual. El más común de los modos que tienen en cuenta esto es CBC.

CBC incluye un mecanismo de retroalimentación, por lo que todos los bloques cifrados están condicionados por el anterior. De esta manera se evita que un atacante pueda sustituir un bloque individual para alterar la información. El funcionamiento de dicho modo se basa en dividir el mensaje en bloques y usar la operación XOR con la cual se combina el cifrado del bloque actual con el del anterior. Para el caso del primer bloque, dado que no tiene bloque anterior, se hace uso de lo que se conoce como un vector de inicialización VI.

Dicho vector proporciona protección frente a los ataques de diccionario. Para ello, es necesario que el vector sea un número aleatorio y que no sea secuencial o predecible. El esquema de cifrado se puede ver en la figura 2.5

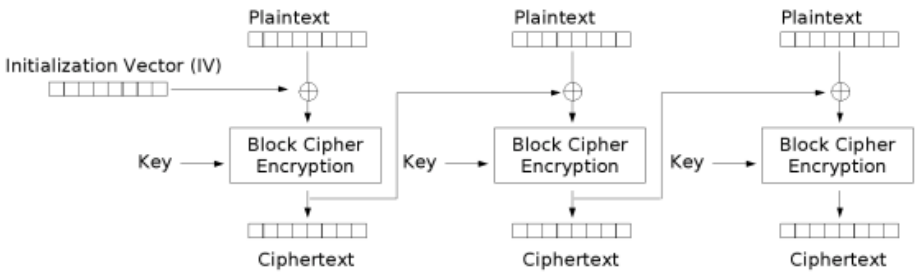

Figura 2.5: Modo de operación CBC (*Block cipher mode of operation*, s.f.).

Una de las desventajas que presenta dicho bloque, es la necesidad de que sea necesario para poder cifrar o descifrar la información disponer de bloques completos. Lo que lleva a la necesidad de utilizar siempre *padding*.

La gestión del *padding* puede introducir algunos inconvenientes que podrían derivar en la posibilidad de inducir ataques como los *padding oracle* (más información de este tipo de ataques en Rizzo y Duong, 2010). Por este motivo, en la práctica se recomiendan otros modos que si permiten realizar un cifrado y descifrado de unidades de información inferiores al tamaño del bloque. Uno de los modos de cifrado con estas características es CTR.

El modo CTR también hace uso de un valor de inicialización, en este caso se suele denominar comúnmente *nonce*. En la práctica puede emplearse un vector de inicialización. Dicho valor de inicialización es combinado con un contador generando así un bloque.

El bloque generado será cifrado mediante un algoritmo simétrico empleando la k correspondiente. Para el siguiente bloque se realizará el cifrado, pero realizando previamente un incremento del contador. Es decir, cada vez que se emplea el algoritmo de cifrado se va variando el valor del *nonce*. Los bloques resultantes cifrados se combinarán cada uno con el bloque de texto plano correspondiente utilizando una operación XOR, dando así finalmente como resultado al bloque cifrado. Dicho proceso puede observarse en la figura 2.6.

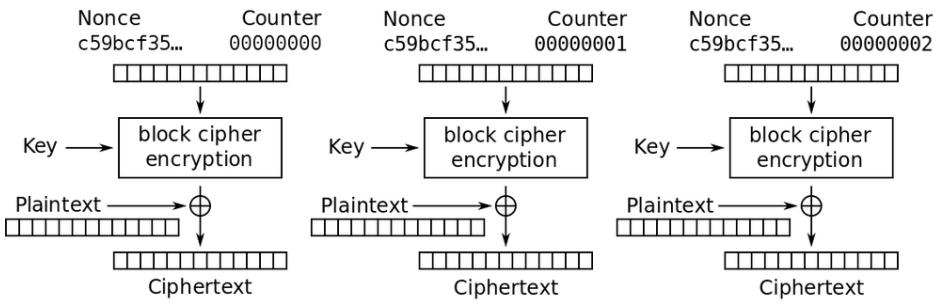

Figura 2.6: Cifrado con el modo de operación CTR (*Block cipher mode of operation*, s.f.).

La aleatoriedad producida por el algoritmo de cifrado, que después es propagada al texto plano con la operación XOR dota de una más que robusta seguridad al bloque cifrado resultante.

El proceso de descifrado es exactamente el mismo, donde de nuevo se emplea el mismo *nonce*, junto a la k. Mediante el algoritmo de cifrado simétrico se vuelven a cifrar ambos obteniendo un bloque cifrado. Finalmente, como en el cifrado se realiza una operación XOR, en este caso entre el bloque cifrado y el criptograma correspondiente, este proceso se puede ver en la figura 2.7. Esto, en cuanto a la implementación del algoritmo, simplifica de manera inmensa su desarrollo.

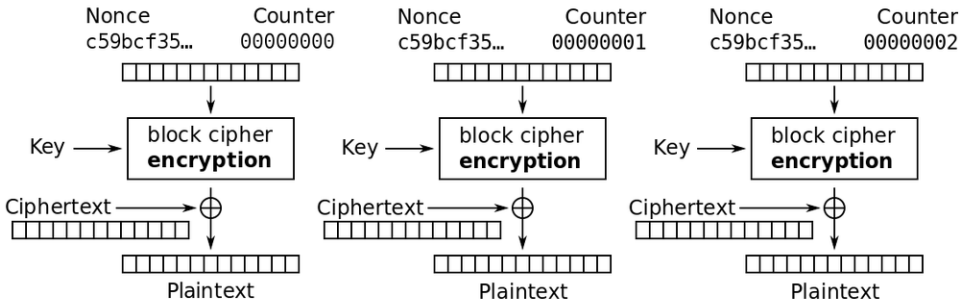

Figura 2.7: Descifrado con el modo de operación CTR (*Block cipher mode of operation*, s.f.).

Es importante destacar que el valor del contador es necesario que sea un número aleatorio usado una única vez en cada cifra para la clave k. Entre las características que presenta este modo se encuentran las siguientes:

- Usando un VI aleatorio y diferente en cada proceso de cifrado, al cifrar dos veces la misma información con la misma clave, se obtendrían criptogramas diferentes.
- Evita que aparezcan perfiles en el criptograma resultante del cifrado.
- Permite efectuar un cifrado y descifrado en paralelo, dado que la entrada de cada bloque al algoritmo de cifrado solamente depende del *nonce*.
- Evita que se produzcan ataques por comienzos y finales iguales.
- Con este modo se simula un cifrador en flujo. Puesto que se produce una secuencia pseudoaleatoria binaria que es combinada con el texto en claro.
- Los errores solo se propagan al bloque en el que se han producido.

A modo de comparación, en la figura 2.8, se puede observar una comparativa de diferentes modos de cifrado, algunos de los más utilizados. Donde, como se puede ver, los dos que dotan de todas las ventajas posibles, son ECB y CTR. Pero como se ha comentado anteriormente, ECB está totalmente desaconsejado, por lo que en definitiva el modo de cifrado más óptimo para elegir es CTR.

ECB	
Electronic codebook	
Encryption parallelizable:	Yes
Decryption parallelizable:	Yes
Random read access:	Yes

CBC	
Cipher block chaining	
Encryption parallelizable:	No
Decryption parallelizable:	Yes
Random read access:	Yes

PCBC (*)	
Propagating cipher block chaining	
Encryption parallelizable:	No
Decryption parallelizable:	No
Random read access:	No

CFB	
Cipher feedback	
Encryption parallelizable:	No
Decryption parallelizable:	Yes
Random read access:	Yes

OFB	
Output feedback	
Encryption parallelizable:	No
Decryption parallelizable:	No
Random read access:	No

CTR	
Counter	
Encryption parallelizable:	Yes
Decryption parallelizable:	Yes
Random read access:	Yes

Figura 2.8: Comparativa de las ventajas en distintos modos de cifrado (Ramió Aguirre, 2020)

2.4 Cifrado Asimétrico

Por otro lado, el otro gran conjunto de algoritmos son los asimétricos. En este tipo de algoritmos se le proporciona a cada usuario dos claves.

Por un lado, la denominada clave pública K_{pub} que como indica su nombre, no requiere ser ocultada del resto de usuarios. Por otro lado, una clave privada K_{priv}.

Esta última es necesario mantenerla secreta, siendo lo más común almacenarla protegida haciendo uso de un algoritmo de cifrado simétrico para garantizar su confidencialidad.

Los algoritmos de cifrado y descifrado son los mismos para ambos procesos, pero es necesario que cada proceso utilice una clave distinta. Por ejemplo, si se cifra con la K_{pub}, el descifrado se hará con la K_{priv} y viceversa, como se puede ver de manera visual en la figura 2.9.

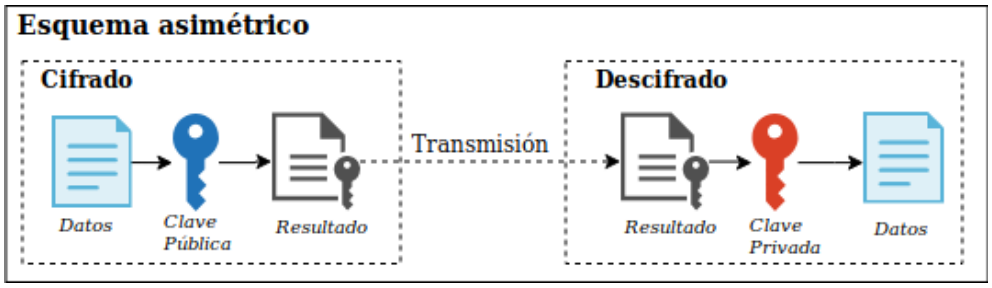

Figura 2.9: Esquema cifrado asimétrico (*Cifrado: ¿simétrico o asimétrico?*, s.f.).

2.4.1 RSA

De entre todos los algoritmos asimétricos, el más popular y utilizado, en gran parte por su facilidad de comprensión y sencillez de implementación, es el algoritmo RSA. Su nombre surge de la combinación de los apellidos de sus tres inventores: Ronald Rivest, Adi Shamir y Leonard Adleman. Aunque en un principio estuvo bajo patente, hoy en día se puede emplear

comercialmente sin problemas.

La gran fortaleza en la que se basa RSA es en la gran dificultad a nivel computación que requiere el problema de factorizar un número compuesto muy grande, que se da como resultado del producto de dos números primos grandes, como indica la siguiente ecuación:

$$n = p * q$$

Para generar un par de claves, lo primero que se necesita es hacer el cálculo anterior, es decir, se escogen dos números primos grandes p y q, para calcular el producto de estos n. A continuación de dicho cálculo se debe escoger, un número natural e, que valide la siguiente condición $0 < e < \phi(n)$ y que además sea a su vez primo con $\phi(n)$, es decir, que cumpla que:

$$mcd(e,\ \phi(n)) = 1$$

La función de Euler $\phi(n)$, representa el número de números naturales menores que n, primos con n.

En relación con ese número e, existe un número natural d que es el inverso de $e\, mod\, \phi(n)$, es decir, que valida la expresión $d\ * e\, mod\, \phi(n) = 1$

Una vez hechos estos cálculos para un usuario, tenemos que:

- La K_{pub} del usuario es el par (n, e).
- La K_{priv} sería el par (n, d).

Las piezas por guardar en secreto serian d y el par p y q. La función de cifrado es la siguiente:

$$c = E_k(m) = m^e (mod\, n)$$

Y la función de descifrado es:

$$m = D_k(m) = c^d (mod\, n)$$

Con esta estructura dada que presenta el algoritmo, un posible atacante del algoritmo se enfrentaría a un problema de factorización para lograr obtener p y q.

Respecto al tamaño de las claves, el $NIST$ recomienda el uso de claves de mínimo 2048 bits, en una publicación especial del NIST (Barker y cols., 2019-03-21). Esto no significa que con los avances tecnológicos que se van realizando, dicho tamaño de clave se quede corto en un periodo de tiempo reducido.

El empleo de claves con longitudes tan superiores a otras como se ha comentado anteriormente en el apartado del algoritmo AES, junto a la complejidad de cálculo implícita en los algoritmos de cifrado asimétrico, repercute en que sean considerablemente más lentos que los algoritmos simétricos.

En la práctica, la utilización de un algoritmo asimétrico como RSA, se hace junto con un algoritmo simétrico como AES. Dándose un esquema donde la información como tal será cifrada con un algoritmo simétrico y será la clave de cifrado de esa información, que por norma general es de un tamaño muchísimo más inferior que la información en sí, la que será cifrada utilizando un algoritmo asimétrico.

Por ejemplo en caso de emplearse RSA. La clave empleada en el cifrado, sería cifrada con la K_{pub} del usuario con el que compartir la información. Una vez el destinatario recibe la información y la clave cifrada, procede a descifrar la clave con su K_{priv} y con esa clave finalmente puede descifrar la información. De esta manera se obtienen los mejores rendimientos en un entorno práctico.

2.4.2 Firma Digital

Los algoritmos asimétricos, a pesar de no ser los algoritmos más óptimos para dotar de confidencialidad a grandes cantidades de información, su funcionalidad no está limitada tan solo a esto, como si ocurre con los simétricos. También permiten garantizar la autenticidad y la integridad de la información. Esta particularidad surge del uso del par de claves en lugar de una única.

El proceso por el cual haciendo uso de un algoritmo asimétrico se puede dotar de autenticidad a cierta información se conoce como firma digital. En esencia, la firma digital resulta ser muy similar a una firma tradicional manuscrita, con la cual la persona firmante da validez y autenticidad a la información que firma.

Esto, trasladado al ámbito digital, se traduce en una cadena de datos que asocia una información digital con el creador de esta. Para que sea igual de válida la firma digital que la tradicional, debe cumplir una serie de propiedades:

- La firma asociada a un documento solo es válida para dicho documento.
- Solo puede ser generada por el titular de la información.
- Cualquiera puede comprobar su verificación, de manera pública.

Es importante resaltar que realizar una firma digital sobre una información no proporciona ningún tipo de confidencialidad sobre esta. En cambio, sí que presenta otras características muy potentes como las siguientes:

- Permite identificar al usuario que ha efectuado la firma de manera inequívoca.
- Garantizar la integridad del documento que ha sido firmado.
- Asegurar el no repudio, con el cual se garantiza que el usuario que hace una firma después no puede afirmar no haberlo hecho.

Como se ha comentado anteriormente, el coste computacional de usar algoritmos asimétricos es demasiado elevado ante grandes cantidades de información. Esto se traduce en que firmar un documento completo, requiere de un tiempo demasiado elevado para hacerlo viable. Es por ello por lo que se busca cifrar una pequeña cantidad de información, o lo que es lo mismo, un resumen, también conocido como un *hash*.

2.4.3 Funciones Hash

Las funciones *hash*, permiten llevar a cabo la transformación de cualquier tipo de archivo o texto, en una serie de caracteres con una longitud fija, sea cual sea el tamaño de la información utilizada como entrada a la función. El resultado de aplicar este tipo de funciones es lo que se conoce como *hash* o resumen.

Dicho resumen representa a los documentos o textos de forma supuestamente única. Un aspecto de estas funciones diferente de los algoritmos vistos anteriormente es que dado que carecen de claves, no son consideradas algoritmos de cifra.

En lo referente al funcionamiento de dichas funciones, este resulta ser realmente sencillo, la información de entrada a la función es dividida en bloques, una vez realizada la división, una fórmula calcula el *hash* para el primer bloque, seguidamente calcula el del siguiente bloque y lo suma al calculado anteriormente. Esto se repite con todos los bloques restantes.

Para garantizar que una función resumen es segura debe cumplir las siguientes características:

- El *hash* obtenido siempre debe tener una longitud fija, sea cual sea la longitud del mensaje de entrada.
- Facilidad de cálculo y rapidez.
- Unidireccionalidad, es decir, dado un resumen $h(m)$, debe ser computacionalmente imposible hallar m a partir del resumen.
- Difusión de bits, un cambio en un bit del mensaje m debería modificar al menos la mitad de los bits del *hash* nuevo en comparación con el anterior.
- Colisión Simple: Dado un mensaje debe ser computacionalmente imposible dar con otro mensaje distinto que dé como resultado el mismo *hash*. Esta propiedad es conocida como resistencia débil a las colisiones.
- Colisión fuerte: Debe ser computacionalmente difícil conseguir un par aleatorio de mensajes diferentes (m, m') que den como resultado el mismo *hash*, $h(m) = h(m')$. Esta última propiedad se conoce como resistencia fuerte a las colisiones.

La seguridad que debe proporcionar un algoritmo *hash*, se encuentra estrechamente ligada a su capacidad de producir un único valor para un conjunto de datos dados.

Una de las familias de funciones *hash* más conocidas es *SHA* (*Secure Hash Algorithm*).

Como se comentó anteriormente, dado el alto coste computacional de la firma digital, es necesario recurrir a estas funciones *hash* con las cuales solamente se firman unas centenas de bits, en comparación con los miles o millones de bytes del documento original.

La generación de una firma digital de un m, utilizando además una función *hash*, sigue los siguientes pasos (se omite el cifrado y descifrado del mensaje para no extenderse, pero debería realizarse también dicho proceso):

- El emisor aplicará una función *hash* al documento, obteniendo el resumen de este, lo que se conoce como $h(m)$.
- Ahora dicho emisor, procede a firmar digitalmente dicho resumen, para ello utilizando su K_{priv} cifra el resumen: $r = E_{K_{priv}}(h(m))$.
- El siguiente paso consiste en enviarle tanto el mensaje m, como la firma r, además de la K_{pub} del propio emisor al receptor.

Este proceso puede verse de forma gráfica en la figura 2.10.

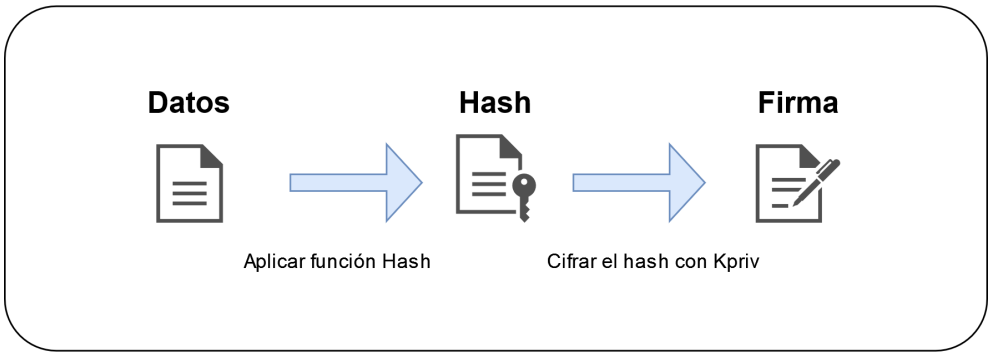

Figura 2.10: Firma digital con función hash

Por el otro lado, el proceso de validación en el otro extremo de la comunicación que puede verse en la figura 2.11 sería el siguiente:

- El receptor descifra r con la K_{pub} del emisor: $h'(m) = D_{K_{pub}}(r)$.
- Después obtendría el *hash* del mensaje m, $h(m)$ y lo compararía con lo obtenido en la anterior operación de descifrado $h'(m) = h(m)$.
- En el caso de coincidir, se puede afirmar que el mensaje enviado, lo ha enviado realmente el usuario al que pertenece dicha K_{pub} y que además dicho mensaje no ha sufrido ningún tipo de alteración.

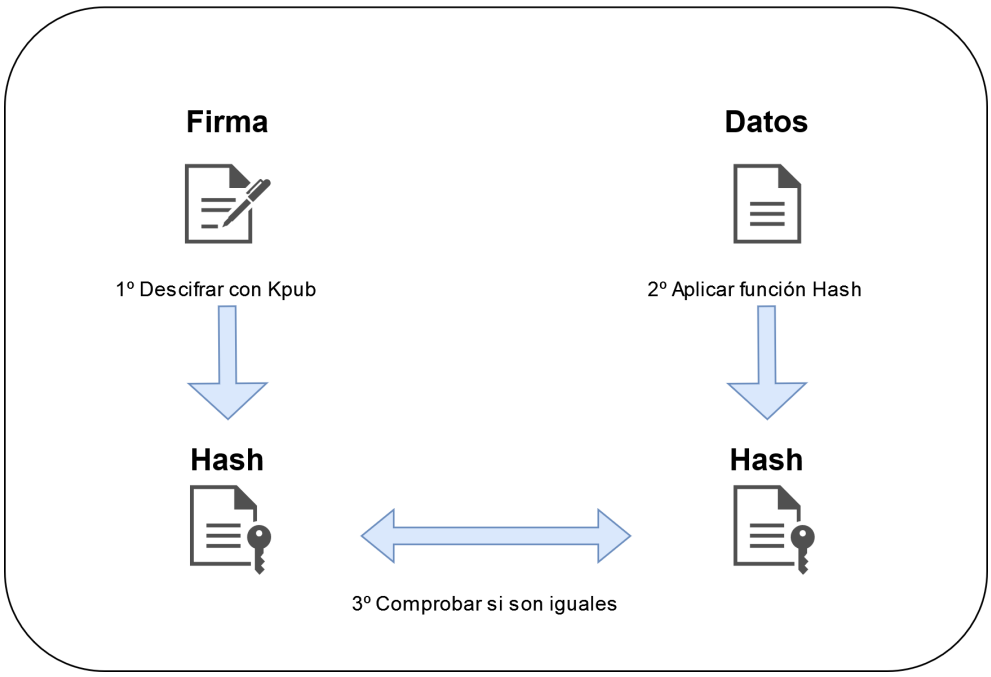

Figura 2.11: Validar firma digital con función hash

2.4.3.1 Derivación de claves

Existen una serie de funciones *hash* criptográficas particulares, conocidas como KDF (Key Derivation Function) que surgieron con la idea de poder transformar una clave de tamaño variable a una clave con un tamaño fijo.

Entre sus usos se encuentra la derivación de claves para obtener nuevas claves, que si cumplan unos requisitos adecuados para ser empleadas en procesos de cifrado. También se utilizan para derivar claves de longitudes diferentes, entre otros usos.

El proceso de derivación fue evolucionando con el tiempo. Hoy en día dichas funciones son conocidas por usarse para garantizar un almacenamiento robusto de las contraseñas de los usuarios.

La utilización sobre una contraseña de un usuario para almacenarla de manera segura va acompañada de un número aleatorio conocido como sal, que es aplicado durante un determinado número de iteraciones. La utilización de esta sal permite evitar ataques de diccionario.

2.4.4 Certificados Digitales

Los algoritmos asimétricos vinieron a resolver el problema que había con la distribución de las claves. Introduciendo el concepto de una clave pública, la cual puede ser compartida abiertamente con el resto de los usuarios, pero es ahí en su propia naturaleza donde surge

uno de los problemas intrínsecos de esta, los ataques MiTM (*Man in the Middle*, para mas información sobre este tipo de ataques, véase INCIBE, 2020-07-16), los que se conocen como ataques de hombre en el medio.

Este tipo de ataques plantean el problema de como se puede estar seguro de que la clave pública que se recibe, es del usuario con el que realmente se quiere realizar la comunicación.

Para solventar este problema surge un concepto conocido como infraestructuras de clave pública (para mas información sobre este concepto, véase Breyha y cols., 2018-12-21), en las que se propone una forma de comprobar la identidad de un usuario, sistema, etc.

Dicha comprobación se basa en el concepto de los certificados digitales, junto con las autoridades de certificación. Estos certificados digitales, crean un vínculo entre el solicitante del certificado y su K_{pub}, junto con un conjunto de información adicional extra a modo de metadatos.

En cuanto a las autoridades de certificación, estas son las encargadas de emitir y firmar los certificados digitales que se soliciten por parte de los usuarios. Para conseguir certificados puede hacerse por distintas vías:

- Solicitando a una autoridad certificadora externa el certificado.
- Utilizando certificados autofirmados.
- Desplegando una autoridad de certificación propia.

Es de vital importancia la protección de la $K_{priv_{AC}}$ de la autoridad de certificación. Con esta se realiza la firma de los certificados que emite, en caso de verse comprometida, se vendría abajo toda la infraestructura de clave pública (PKI).

Todos los procesos de comprobación de validez de un certificado se ejecutan de manera transparente al usuario, exceptuando casos de error, como que el certificado no sea válido. En estos casos se le notifica al usuario del error.

El formato más extendido en la actualidad para los certificados, es el formato $X.509$, descrito en la RFC 5820 (Boeyen y cols., 2008). Dicho estándar no está atado a ningún algoritmo en particular y en este podemos encontramos una serie de campos:

- La versión de $X.509$ empleada.
- El número de serie del certificado.
- El periodo de validez.
- La clave pública del sujeto.
- El algoritmo empleado para la firma digital.
- Otros campos.

- La firma digital de todos los campos del certificado.

Toda autoridad certificadora dispone de un par de claves:

- La $K_{priv_{AC}}$ con la cual la AC puede firmar los certificados.
- La $K_{pub_{AC}}$ utilizada por los usuarios para comprobar la validez de un certificado expedido por una AC.

Para generar un certificado C_A para un usuario A, la autoridad certificadora realiza lo siguiente:

$$C_A = E_{K_{priv_{AC}}}(K_{pub_A})$$

Como se puede observar, es necesario que el usuario, en este caso A envie su K_{pub_A} a la autoridad certificadora para que esta genere el certificado. El proceso de autenticación del certificado por parte de un usuario, B, requiere solicitar la K_{pub_A} y además pedir a la AC el C_A. Con estos dos se valida que la K_{pub_A} es la misma que aparece en el certificado, tal que así:

$$K_{pub_A} = D_{K_{pub_{AC}}}(C_A)$$

Sobre estos dos procesos, añadir que es necesario en todo momento confiar en la $K_{pub_{AC}}$, dado que es la que emplean los usuarios para afirmar que un certificado de otro usuario es válido.

2.5 Comunicaciones seguras

Existen una serie de técnicas que permiten dotar de mayor seguridad a una comunicación entre un emisor y un receptor. Algunas de las más destacas son las que se detallan en las siguientes secciones.

2.5.1 TLS

Los certificados digitales, se han trasladado también a la seguridad en las comunicaciones. Utilizándose en protocolos como TLS (*Transport Layer Security*, para más información, véase Priego García, 2018-06-03). Dicho protocolo tiene distintos usos, donde uno de los más extendidos y conocidos es en la protección del tráfico HTTP.

Cuando se realiza una protección del tráfico HTTP con TLS, el protocolo HTTP pasa a denominarse HTTPS. Con HTTPS se logra dotar a las comunicaciones de la garantía de que en caso de que haya un sniffer (para mas información sobre este tipo de herramientas, consúltese Avast, 2022-01-13), el tráfico capturado se encuentre totalmente cifrado para el atacante.

TLS es un protocolo de seguridad criptográfico que se utiliza para garantizar la seguridad en las comunicaciones. Dispone de protección de la confidencialidad de los datos intercambiados gracias a la utilización de algoritmos de cifrado simétricos. Las claves que se emplean en estos son previamente acordadas haciendo uso de diferentes métodos, entre ellos algoritmos asimétricos como RSA.

En la figura 2.12 se observa el procedimiento de establecimiento de una comunicación segura entre un cliente y un servidor, haciendo uso de certificados digitales. El establecimiento de la conexión segura conlleva los siguientes pasos:

1. El cliente (navegador web en la figura) informa al servidor que quiere establecer una comunicación segura. Para ello, le indica la versión del protocolo TLS que soporta junto a otros parámetros de la comunicación.

2. Si el servidor soporta TLS, se lo indicara al cliente junto con su certificado digital.

3. El cliente valida el certificado digital del servidor.

4. Una vez realizadas las validaciones, el cliente envía la clave que se empleara en el cifrado simétrico de los datos de la comunicación, Dicha clave estará cifrada con la K_{pub} del servidor, que viene adjunta en el certificado.

5. A partir de este momento, se empleará la clave enviada para cifrar todos los datos con un algoritmo simétrico.

De manera opcional también se pueden establecer comunicaciones en las que se requiera que el cliente también presente un certificado digital que validara el servidor.

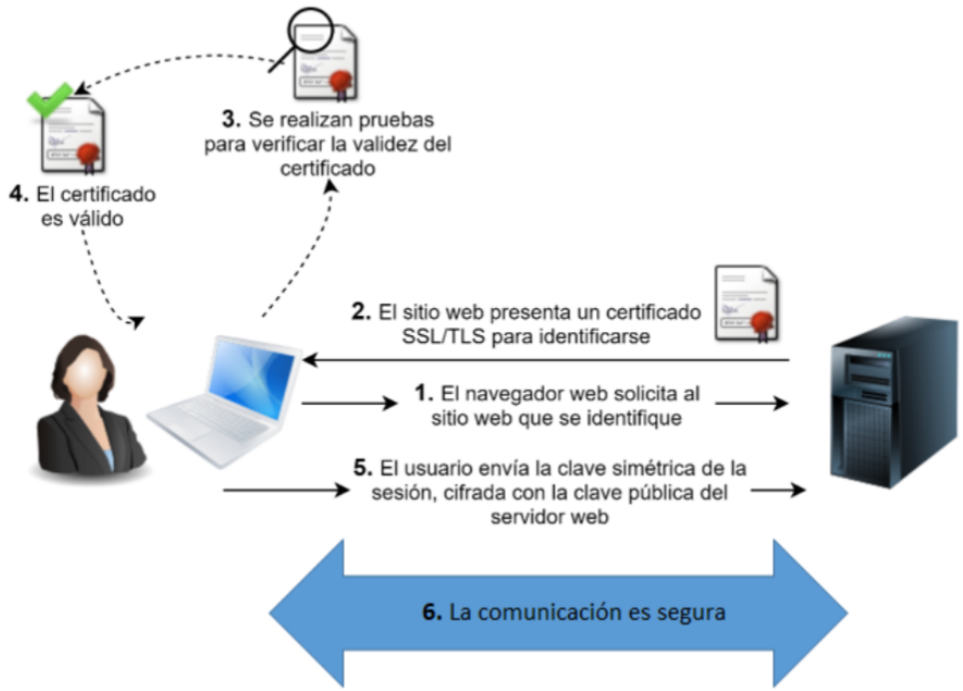

Figura 2.12: Esquema TLS (CCN, 2020)

2.5.2 JWT

Otra forma de dotar de mayor seguridad a las comunicaciones es mediante el uso de tokens, en concreto existe un estándar abierto definido por la RFC 7519 (Jones y cols., 2015), denominado *JSON Web Token* (JWT). El cual permite dotar de un mecanismo de autorización a una comunicación.

El estándar permite generar una serie de *tokens*, que sirven entre otras cosas para validar la autorización de los usuarios sobre un servidor. El proceso de generación de un *token* en un servidor es el siguiente:

1. El usuario inicia sesión en el servidor.
2. El servidor genera un *token* y se lo envía al cliente.
3. El cliente adjunta el *token* en la cabecera de las peticiones que realice al servidor.
4. El servidor valida el *token* y permite el acceso al usuario. En caso de no ser válido, se le deniega el acceso.

Un JWT está formado por tres partes separadas y muy diferencias codificadas en base64. Cada una de estas partes se separan por un punto, tal que así: cabecera.carga.firma

- La cabecera del token: Consiste típicamente en dos partes. Por un lado, el tipo de token, en caso de usar JWT, ese será el tipo. Por otro lado, el algoritmo que se ha utilizado para firmar el token (HMAC, RSA, SHA256).
- La carga del token (Payload): Contiene todas las propiedades que se quieran especificar sobre el token, habiendo una serie de propiedades definidas en el estándar. La información se añade como pares clave/valor que se conocen como *claims*. Uno de los campos más destacables es el tiempo de expiración del token, que se emplea para saber cuándo un token deja de ser válido.
- Firma: La firma resultante de aplicar el algoritmo de firma que se especificó en la cabecera del token sobre los dos campos anteriores (la cabecera y la carga del token).

Es importante resaltar, que los tokens no están cifrados. Toda la información que hay en estos se puede simplemente decodificar y será visible. En la figura 2.13 se puede ver como la información que añadimos en un token es totalmente visible.

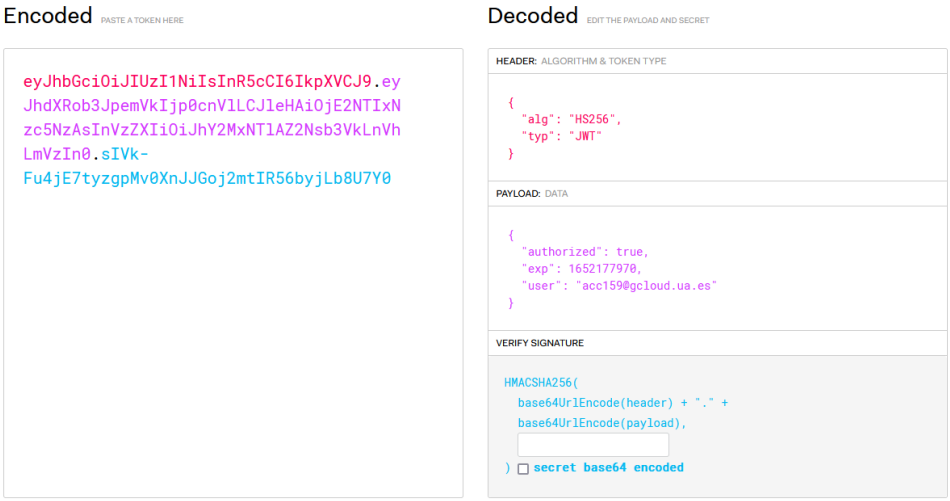

Figura 2.13: Ejemplo JWT usado en el proyecto

2.6 Gestión de tareas y las herramientas de gestión de tareas

La gestión de tareas es uno de los procesos más relevantes que hay que tener en cuenta sea cual sea el ámbito en el que se aplique. Ya sea en el mundo empresarial o personal, reporta múltiples beneficios como:

- Lograr una mayor optimización del tiempo.
- Aumentar la productividad y la efectividad.
- Mejorar la capacidad de tomar decisiones.

Por estos y otros motivos, sea cual sea el ámbito en el que se encuentre un usuario, ya sea en el personal o empresarial, tanto para empresas con personal muy reducido como para grandes corporaciones, es necesario disponer de herramientas de gestión de tareas que nos permitan realizar dicha gestión de manera óptima y accesible.

Pero no solo basta con que este tipo de software proporcione estas características. Puesto que una tarea como tal puede oscilar desde algo muy sencillo y simple, hasta poder contener datos de carácter personal o incluso sensibles de una persona u organización.

Por ello, hoy en día resulta vital que cualquier software de gestión de tareas garantice a los usuarios de este, la confidencialidad de la información que se utiliza en estos.

Además, en un mundo interconectado como el actual, donde en la mayoría de los casos se trabaja en proyectos colaborativos donde intervienen distintas personas, también resulta vital que se garantice la autenticidad e integridad de la información.

Todos estos aspectos aplicados a un software dan lugar a que el mismo pueda ser considerado como un sistema robusto y fiable, logrando trasladar la seguridad de este a los usuarios.

2.6.1 Soluciones actuales

En el ámbito de la gestión de tareas, son innumerables las diferentes aplicaciones y soluciones que existen. Cada una aporta diferentes formas y visiones sobre como poder realizar una correcta gestión de las tareas, algunas son más simples y fáciles de usar y otras más complejas para un usuario sin conocimientos específicos.

Algunas de las más conocidas son por ejemplo Trello (Trello, s.f.-a), Jira (Jira, s.f.-b) o ClickUP (ClickUp, s.f.-b).

Dejando a un lado sus funcionalidades en cuanto a gestión de tareas y centrando la vista en el apartado de la seguridad en estas. En cuanto al cifrado de la información, todas comparten el mismo procedimiento.

Establecen una conexión segura mediante TLS con los servidores correspondientes, a través de la cual efectúan lo que se denomina un cifrado en tránsito por donde envían los datos. Y una vez llegan al servidor es cuando se les realiza lo que se conoce como un cifrado en reposo mediante el algoritmo de cifrado AES.

Todas se centran tan solo en garantizar la confidencialidad de la información. No emplean otro tipo de técnicas o algoritmos para proporcionar otros aspectos como la integridad, autenticidad y no repudio de la información (Trello, s.f.-b; ClickUp, s.f.-a; Jira, s.f.-a).

3 Objetivos

El alcance del proyecto puede verse reflejado en los siguientes objetivos:

1. Análisis e implementación de los algoritmos y técnicas criptográficas:

   El objetivo base sobre el que surge dicho proyecto, es la búsqueda, análisis e implementación de diferentes algoritmos y técnicas criptográficas con el fin de dotar de una capa de seguridad robusta a un sistema informático.

2. Software Cliente/Servidor para la gestión de tareas en proyectos de manera sencilla:

   Se plantea el desarrollo de un software cliente/servidor que permita a cualquier usuario, sea cual sea su conocimiento sobre informática y gestión de tareas, el poder realizar dicha gestión de manera cómoda y sencilla. Implementando una interfaz accesible.

3. Protocolos de comunicación seguros:

   Para garantizar que todas las comunicaciones entre los diferentes clientes con el servidor se efectúen de manera segura, se utilizaran protocolos de comunicación seguros que nos garanticen que terceras partes no puedan acceder a la información entre un cliente y el servidor.

4. Cifrado de la información:

   Se emplearán diferentes algoritmos criptográficos tanto de carácter simétrico como asimétrico para poder efectuar un cifrado de la información, pudiendo garantizar así la confidencialidad de esta, en un esquema donde la misma, está siendo compartida con diversos usuarios.

5. Utilización de la firma digital sobre los elementos y eventos vinculados a las tareas:

   Se empleará dicha técnica criptográfica para dar integridad y autenticación tanto, por un lado, a todo tipo de elementos que se adjunten a una tarea, como los diferentes eventos que sucedan sobre esta.

6. Empleo de certificados digitales para respaldar la firma digital:

   Se desplegará una autoridad certificadora con la cual poder generar certificados digitales para cada uno de los usuarios del sistema con el fin de dar mayor validez al sistema de firmas.

7. Sistema de roles para controlar accesos:

   Implementación de un sistema de roles con el cual poder controlar y limitar las acciones de los usuarios sobre los distintos elementos que componen el sistema.

4 Planificación

En este capítulo se detallará la metodología empleada para desarrollar con éxito el proyecto junto a otros aspectos como la gestión de tareas y el control de versiones realizado.

4.1 Metodología

En cuanto a la metodología seguida para el adecuado desarrollo del proyecto, se ha escogido una metodología de carácter ágil. Este conjunto de técnicas permiten gestionar proyectos software en los cuales los requisitos van evolucionando durante el desarrollo, para ir adaptándose a las necesidades que surgen.

Todas las metodologías que se basan en este marco de trabajo ágil comparten distintas características como:

- Considerar a los individuos e interacciones de manera más importante que a los procesos o herramientas empleadas.
- El funcionamiento del software es prioritario.
- Estrecha colaboración con el cliente del proyecto.
- Priorizan dar respuesta a los cambios que surjan durante el desarrollo frente a seguir un plan estricto.

De entre las diversas metodologías que siguen este esquema ágil, para la realización de este proyecto se ha optado por enfocarse sobre la metodología XP (*Extreme Programming*, para mas información sobre la metodología, véase Bahit, 2016). Algunas de las características más relevantes de esta, que se ven reflejadas en el desarrollo que se ha realizado, son:

- Metodología centrada en la programación del producto, el enfoque ha sido poder llevar a cabo el desarrollo de un sistema utilizable por los usuarios con las medidas de seguridad planteadas aplicadas.
- El cliente es un elemento fundamental en el proyecto, puesto que su opinión debe ser conocida en tiempo real, por ello se da una alta interacción del equipo de desarrollo con el cliente (tutor en este caso), con el cual se van haciendo reuniones en reducidos intervalos de tiempo. La duración de las iteraciones ha sido normalmente de unas 2 semanas, con alguna excepción de 3 semanas máximo.
- Permitir modificaciones en las tareas, aunque ya se hayan llevado a cabo las tareas, son susceptibles de ir presentando cambios que se irán efectuando. Esto ha sucedido con algunas funcionalidades cuyo enfoque se planteó de una manera y con el desarrollo del

proyecto se han observado que debían ser modificadas para encajar mejor en el enfoque final del proyecto.

Para llevar a cabo una adecuada implementación del proyecto se ha optado por estructurarlo en una serie de hitos a nivel general, para disponer de una referencia general sobre las diferentes fases por las que avanzar en el desarrollo del proyecto.

En cada uno de los hitos se ha ido trabajando en pequeñas mejoras una tras otra. Una vez se programaba una funcionalidad se realizaban una serie de pruebas para darla como válida, en caso de darse errores, se continuaba trabajando en esta. De lo contrario, se procedía a desarrollar la siguiente funcionalidad prevista, como recomienda la metodología XP. Los hitos generales sobre los que se han trabajado son:

1. Desarrollo API REST junto a la integración con la base de datos y pruebas del API.
2. Integración con el cliente.
3. Cifrado de la información.
4. Funcionalidades generales.
5. Capa de seguridad adicional.
6. Interfaz visual.

Como se ha comentado, se han efectuado reuniones durante reducidos periodos de tiempo con el tutor del proyecto. En las cuales se validaban los objetivos planteados en la iteración anterior, en algunos casos se sugerían modificaciones necesarias para que el proyecto se desarrollara con éxito siguiendo los objetivos iniciales. Finalmente, se definían los hitos de la siguiente iteración.

Para la realización de estas reuniones, dada la situación actual y por motivos de accesibilidad, estas se han efectuado por medio de videollamadas empleando la herramienta Google Meets (*Software Google Meet*, s.f.).

4.2 Control de versiones

De cara al desarrollo del sistema se ha utilizado el software de control de versiones Git (*Software Git*, s.f.), empleado durante una gran cantidad de asignaturas relacionadas con el ámbito de la programación durante el transcurso del grado realizado.

Git se emplea para poder ir almacenando los diferentes cambios y mejoras que se han ido efectuando con el paso del tiempo en el proyecto. Para el almacenamiento de las diferentes versiones se ha empleado la plataforma GitHub (*Software Github*, s.f.), que facilita en gran medida el proceso.

El procedimiento seguido se ha basado en usar una rama diferente a la principal para poder ir trabajando en las funcionalidades y mejoras del proyecto. Una vez desarrolladas todas, se

han trasladado a la rama principal. En la figura 4.1 se puede ver la rama de desarrollo que se ha empleado.

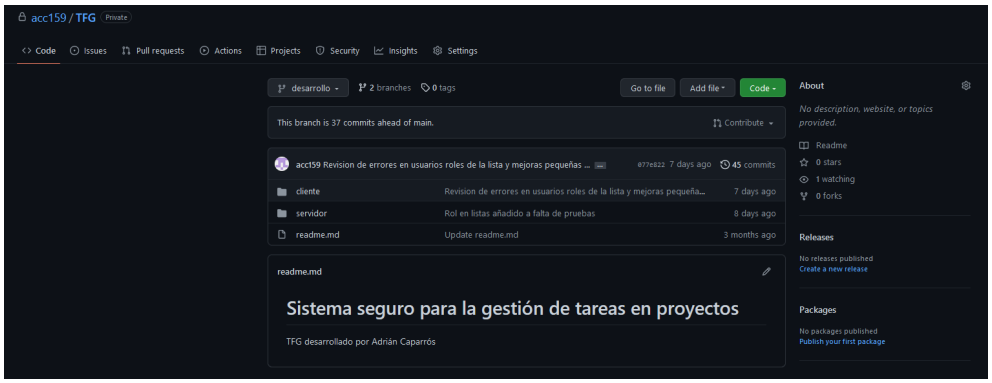

Figura 4.1: Github del proyecto

A la hora de trabajar en el proyecto, una vez finalizadas las tareas en las que se estaba trabajando, se actualizaba la versión del software en el repositorio. De esta manera, en caso de darse algún problema con el código, poder tener un respaldo en diferentes puntos de este.

En ningún momento se ha llegado a necesitar volver a una versión anterior durante el desarrollo de este proyecto, pero hacer uso de este tipo de herramientas siempre dota de gran seguridad a un programador durante el desarrollo de un software.

4.3 Gestión de las tareas

Para llevar a cabo la gestión de las tareas que forman el proyecto que se ha desarrollado, se ha empleado la herramienta Trello, mencionada anteriormente en las soluciones actuales 2.6.1.

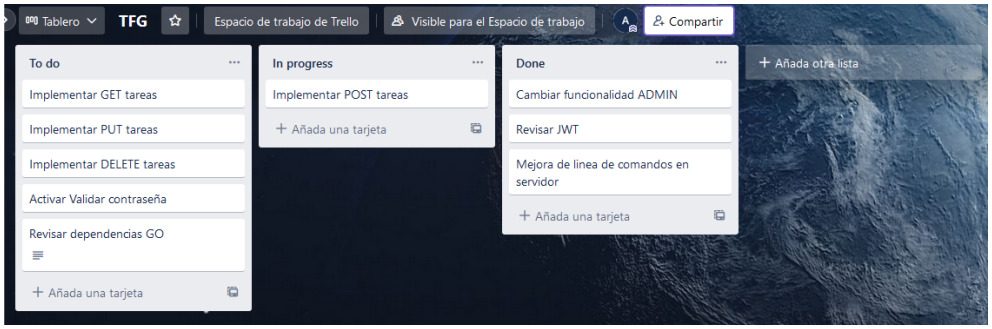

Figura 4.2: Trello del proyecto

Su elección se debe a que proporciona al usuario poder realizar una gestión de tareas de manera sencilla y accesible, gracias la sencillez de su interfaz gráfica. Se han utilizado lo que se conocen en Trello como listas dentro de un tablero, donde, por un lado, encontramos la lista de las tareas pendientes (*To do*), las que están en progreso (*In progress*) y por último las ya finalizadas (*Done*) como se puede ver en la figura 4.2.

5 Desarrollo

A continuación se van a detallar los diferentes aspectos relevantes al desarrollo del proyecto. Además, se detallarán las técnicas y algoritmos criptográficos implementados en el sistema desarrollado.

5.1 Arquitectura

De cara al desarrollo del sistema, se ha implementado una API REST con una arquitectura cliente/servidor. Donde estos han sido concebidos para realizar u ofrecer una serie de funcionalidades muy diferenciadas.

En el lado del servidor se ha desplegado una API REST, donde por medio de métodos como GET, POST, PUT o DELETE, se efectúan las operaciones pertinentes sobre los datos del sistema. Dichos datos se trabajan en formato JSON o XML. En este proyecto se ha optado por elegir JSON, dado que es un formato fácil de entender y de trabajar.

La elección de desarrollar una API REST, proporciona una serie de ventajas como:

- Disponer de una interfaz uniforme que permita que todos los clientes, mediante HTTP, puedan hacer uso de esta. Independientemente de la tecnología que utilice el cliente.
- Otorga un alto nivel de escalabilidad.
- Facilidad de desarrollar de manera independiente cliente y servidor.
- Comodidad de cara a introducir mejora en un futuro para extender el uso de la API.

La elección de la arquitectura cliente/servidor está centrada en que el empleo del sistema sea simultáneo por distintos clientes que hagan peticiones a un servidor centralizado. Para la comunicación entre cada uno de estos clientes con el servidor, se hará uso del protocolo HTTPS, para garantizar la seguridad en las comunicaciones.

Las funcionalidades a rasgos generales del servidor y del cliente son:

- Servidor: Recibe y procesa las peticiones de los clientes a la API. Además de servir de intermediario con la base de datos, para poder dar persistencia a la información del sistema. También es empleado para servir como autoridad certificadora.
- Cliente: Se encarga de la representación visual de la información del sistema para el usuario. Implementa las diferentes técnicas y algoritmos criptográficos que garanticen la confidencialidad, autenticidad e integridad de la información. Realiza las llamadas pertinentes a la API del servidor.

Respecto al servidor, comentar que por simplificación de cara al desarrollo y al no tratarse de un entorno en producción, se optó por utilizar el propio servidor como autoridad certificadora. En un entorno en producción se separaría la autoridad certificadora en otro servidor diferente.

En el apartado 5.5 se comentan con detalles las funcionalidades que efectúa cada uno. En la figura 5.1 se puede visualizar la arquitectura del sistema con sus diferentes elementos.

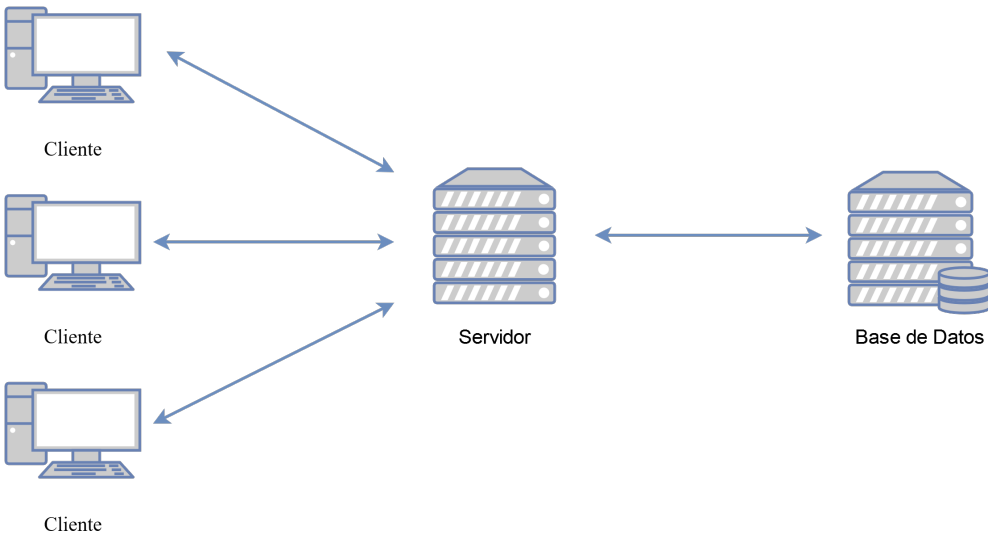

Figura 5.1: Arquitectura del sistema desarrollado

5.2 Tecnologías empleadas

Para el desarrollo del proyecto, se han usado distintas tecnologías y herramientas. Algunas de ellas forman parte directamente del proyecto. Otras han servido de gran ayuda para llevar a cabo de manera satisfactoria el desarrollo de este.

5.2.1 Golang

Comúnmente conocido como GO (*Lenguaje de programación GO*, s.f.), es un lenguaje de programación compilado creado por Google. Se ha elegido como el núcleo fundamental para el desarrollo del sistema tanto en la parte del cliente como en el servidor. Algunas de las ventajas del lenguaje son:

- Lenguaje sumamente rápido.
- Es multiplataforma, pues puede ser ejecutado tanto en Windows, Mac, Linux, etc.
- Dispone de muy buenas librerías de carácter criptográfico.

Esta última ventaja, sumada a la gran potencia del lenguaje, han sido los factores que han decantado que se opte por dicho lenguaje de programación.

5.2.2 html/template

Se ha empleado un paquete de GO conocido como *html/template* (*Paquete html/template de GO*, s.f.), que permite el uso de plantillas en GO. Estas plantillas básicamente permiten cargar código HTML realizando un renderizado del mismo. Además, permiten que se les pueda pasar todo tipo de estructuras de datos de GO a la hora de que se renderice la plantilla que se quiera cargar.

El empleo de las plantillas otorgan una gran ventaja, ya que disponen de un escapado de los datos automático. De esta manera nos protegen de ataques del tipo XSS (*Cross − site scripting*, para conocer más sobre este tipo de ataques, véase Jeremiah y cols., 2007), puesto que desde GO a la hora de procesar el HTML se escapan todos los caracteres antes de ser mostrados.

5.2.3 Lorca

Es una librería de GO (*Librería Lorca*, s.f.), la cual permite desarrollar aplicaciones de escritorio haciendo uso de tecnologías de desarrollo web como son HTML, CSS, etc. Aprovechando así, las capacidades de estas para poder desarrollar interfaces visuales agradables y sencillas para el usuario.

La librería permite realizar un proceso conocido como *bind* por el cual desde código JavaScript se podrán realizar llamadas a funciones en GO previamente declaradas con *bind*. Estas funciones en GO harán las operaciones pertinentes y devolverán los resultados a JavaScript para que puedan ser mostrados a los usuarios. Este proceso y el uso de las plantillas *html/template* se comentan en detalle en la sección 5.5.2.

Requiere de tener instalado Google Chrome en el equipo donde se utilice.

5.2.4 Javascript, HTML, CSS, Bootstrap, Jquery

De cara a la parte de la interfaz visual, se han usado los siguientes lenguajes y tecnologías:

- El lenguaje de marcado HTML.
- El lenguaje de estilo CSS.
- Para dotar de funcionalidad particular a la interfaz y para el proceso comentado anteriormente se ha empleado *JavaScript*.
- También se ha empleado la tecnología *Bootstrap* (*Framework Bootstrap*, s.f.) para mejorar el estilo de los elementos dispuestos en pantalla.
- Y por último *jQuery* (*Librería jQuery*, s.f.), para alguna funcionalidad muy particular.

5.2.5 MongoDB

De cara al sistema de base de datos elegido, se ha optado por usar un sistema de base de datos *NoSQL*. El motivo de dicha elección se detalla en el apartado de 5.3. De las tecnologías de base de datos *NoSQL* existentes se ha optado por elegir MongoDB (*MongoDB*, s.f.) para el proyecto.

MongoDB es un sistema de base de datos *NoSQL* que está orientado a documentos. Es un proyecto de código abierto. Destaca dentro del mundo *NoSQL*, debido a su gran capacidad de aprovechamiento de los diferentes recursos de la máquina en la que se encuentra.

MongoDB no emplea tablas como en las bases de datos *SQL*, es una base de datos orientada a documentos. Dichos documentos son almacenados en un formato conocido como BSON. Este es un formato de serialización, que resulta ser simplemente una codificación binaria de documentos similares a JSON.

Se ha optado además por MongoDB dado que dispone de drivers para utilizar en la gran mayoría de los lenguajes de programación, junto a una documentación muy buena. Además, puede ser usado tanto en local como en *cloud* con MongoDB Atlas (*MongoDB Atlas*, s.f.). Para el desarrollo del proyecto se ha optado por instalar una versión local de MongoDB, pero también se podría emplear la versión *cloud*.

5.2.6 MongoDBCompass

Para la facilitación de cara a poder ir consultando la información sobre la que se trabajaba en el sistema, se ha empleado la herramienta *MongoDBCompass* (*Software MongoDB Compass*, s.f.). Esta permite poder visualizar, editar y eliminar los datos que hay en la base de datos de manera muy sencilla y rápida.

5.2.7 Visual Studio Code

De cara a la edición del código fuente de todo el proyecto, se ha empleado el editor *Visual Studio Code* de Microsoft (Microsoft, s.f.), el cual facilita muchísimo la tarea de programar, permitiendo realizar una depuración sencilla y rápida del código. También permite tener diferentes consolas abiertas en este para hacer pruebas simultáneas. Y se le pueden incluir diferentes extensiones que facilitan y permiten programar de manera óptima y eficiente.

5.2.8 Postman

De cara a la construcción y principalmente prueba de la API desarrollada en el servidor, el software *Postman* (*Software Postman*, s.f.) ha sido un elemento de gran ayuda. Se ha utilizado para poder ir probando los diferentes *endpoints* de la API a medida que se han ido desarrollando.

5.3 Base de datos

La base de datos es un elemento fundamental en cualquier sistema informático. Con esta se logra dar persistencia a los datos con los que interactúan los usuarios. Existen distintos tipos de bases de datos, en este proyecto en particular, se ha optado por elegir una base de datos no relacional ($NoSQL$), frente a las clásicas relacionales.

La gran particularidad de las bases de datos no relacionales se da en su propio nombre, dado que desaparecen por completo las relaciones sobre las que se basan las relacionales. Además, en estas no se emplea el lenguaje SQL como herramienta de búsqueda, sino solo como un apoyo a las búsquedas.

La diferencia de un tipo a otro no solamente se aprecia en la desaparición de las relaciones, sino que directamente la estructura de tablas sobre la que se basan las relacionales también desaparece. Las no relaciones no trabajan con estructuras definidas, lo que permite crear esquemas de la información de un sistema totalmente flexibles. Además, disponen de una gran escalabilidad y están pensadas para la gestión de grandes volúmenes de datos.

Existen diferentes tipos de bases de datos $NoSQL$, desde las más utilizadas, las documentales, pasando por las de tipo clave-valor, hasta las que emplean grafos o columnas.

Las que son documentales almacenan, como su nombre indica, la información como documentos. Emplean estructuras simples como JSON o XML, con un identificador único para cada nueva entrada. Además, dotan al programador de una gran versatilidad para su utilización en multitud de escenarios.

Se ha escogido una base de datos no relacional, dado que en el proyecto esta solo es empleada como una herramienta para dar persistencia a la información. En el lado del servidor, toda la información recibida lo hace totalmente cifrada, es decir, campos como la fecha de una tarea, se almacenan cifrados en la base de datos. Esto conlleva a que en ningún momento se realicen búsquedas por campos específicos de las estructuras de datos (fecha de una tarea, nombre de un proyecto, etc.) o incluso poder relacionarlos entre ellos. Por este motivo, de usarse una base de datos relacional, se estaría desperdiciando gran parte de su potencia. Y dada la gran velocidad y rendimiento que presentan las $NoSQL$, se ha optado finalmente por una de este tipo.

En este tipo de base de datos la información se divide en colecciones. Estas sirven para agrupar documentos que comparten características similares. Aunque no es necesario que todos los documentos de una colección presenten los mismos campos.

5.3.1 Colecciones del proyecto

En el proyecto en particular se trabajan con un total de 6 colecciones. En cada una se añadirán los documentos que le correspondan, con una estructura que se define a nivel de código cuando se va a insertar el propio documento. Las colecciones del sistema son las siguientes:

- Una para los usuarios del sistema *users*.
- Para almacenar los proyectos *proyects*.
- Otra para las listas *lists*.
- Y otra para las tareas *tasks*.
- Una necesaria para guardar las relaciones de los usuarios con los proyectos y listas *relations*. También almacena las claves empleadas en el cifrado de los proyectos y listas. Esto se detalla en el apartado 5.6.
- Y una última para poder almacenar la traza de los eventos que ocurren en el sistema *logs*.

Como se puede apreciar, el sistema presenta tres estructuras con las que trabajaran los usuarios, por un lado, la estructura principal son los proyectos. Dentro de estos se podrán añadir listas asociadas a ese proyecto concreto. Y por último, dentro de cada lista se añadirán las tareas correspondientes a esta. Dicha organización puede verse de manera visual en la figura 5.2.

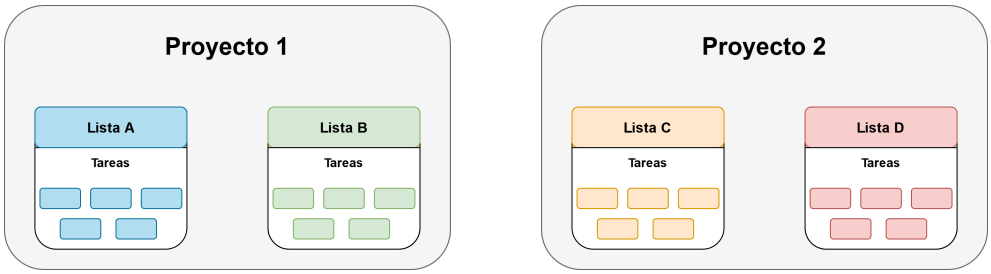

Figura 5.2: Esquema organización de los elementos del sistema

5.3.2 Índices

Una de las funcionalidades que ofrece MongoDB, es el poder aplicar lo que se conocen como índices. Estos son una estructura de datos especial, que almacena una pequeña porción de los datos de una colección. A la hora de recorrer todos los documentos de una colección, en caso de existir estos índices, el proceso es más eficiente y rápido.

Por defecto, todas las colecciones incorporan un índice para el campo del identificador único, que tiene cada documento una vez es añadido a una colección. Además de dichos índices por defecto, en el proyecto se han añadido una serie de índices para proporcionar mayor velocidad a la recuperación de los datos más solicitados.

En la colección de usuarios, se ha añadido un índice sobre el campo *email* de cada usuario. Dado que se emplea de manera continua para recuperar los documentos deseados de la colección.

En la colección de tareas, se ha añadido uno para el campo *listID* que sirve para identificar a que lista pertenece cada tarea. De esta manera se optimiza el tiempo de recuperar las tareas de una lista concreta.

Y por último en la colección de relaciones se ha implementado un índice compuesto. Es decir, en este caso el índice está formado por los campos que representan el *email* del usuario y el *id* del proyecto. Puesto que constantemente se requiere recuperar las relaciones de cada usuario con un proyecto o con las listas de un proyecto concreto.

Al iniciar el servidor la primera vez, las tres colecciones con los índices personalizados mencionados anteriormente son creados automáticamente. El resto de colecciones se originan automáticamente cuando se introducen los primeros documentos en estas. En caso de ya existir las tres colecciones anteriores al desplegar el servidor, simplemente no sucede nada y se trabaja con las colecciones ya existentes.

5.4 Tipos de usuarios

En lo referente a los tipos de usuarios que intervienen en el sistema, para evitar mayor complejidad se ha optado por definir un único usuario (*admin*) con unas características muy concretas. El resto de usuarios poseen todos las mismas funcionalidades y características.

El usuario *admin* presenta una única funcionalidad, la capacidad de habilitar o deshabilitar las cuentas de los usuarios que utilizan el sistema. Cuando se inserta un documento que representa a un usuario del sistema en la base de datos, se le incorpora un campo *status*. A través de dicho campo, el usuario *admin* puede habilitar o deshabilitar a un usuario.

El usuario *admin* debe ser creado el primero en el sistema de manera obligatoria. En el panel de registro de la interfaz del cliente a la hora de realizar el registro de un usuario, si se intenta registrar primero otro usuario diferente se le indicara que esto no es posible sin previamente generar al usuario *admin*. En caso de existir el usuario en la base de datos ya no es necesario volver a generarlo.

Una vez inicie sesión este usuario, se le mostrará un panel con el que comprobar el estado de todos los usuarios del sistema y poder modificar dicho estado. El panel de administrador se puede observar en la figura 5.3.

Figura 5.3: Panel del usuario admin

Se ha limitado su funcionalidad tan solo a esto, en lugar de otorgarle acceso a todos los proyectos, listas y tareas del sistema. Dado que en ese caso, se abriría una brecha de segu-

ridad para el sistema, pues de vulnerar dicha cuenta de usuario se podría acceder a toda la información del sistema. El desarrollo del sistema ha sido planteado para garantizar la máxima privacidad de la información y únicamente permitir el acceso a esta, a las personas autorizadas.

El resto de los usuarios, como se ha indicado, son considerados totalmente iguales y disponen todos de las mismas capacidades. Aunque, como se detalla en 5.6.6, dentro de cada estructura de datos se implementa una política de roles, para controlar las acciones de los usuarios sobre estas.

5.5 Funcionalidades

Como se comentó anteriormente, tanto el cliente como el servidor realizan una serie de funcionalidades diferentes. A continuación se detallarán dichas funcionalidades, sin entrar en detalles sobre la seguridad del sistema. Los aspectos y funcionalidades relacionados con la seguridad se detallan en el apartado 5.6

5.5.1 Servidor

La idea principal sobre la que se ha desarrollado el servidor es para el despliegue de una API REST en este. Dicha API se encarga de atender las peticiones efectuadas desde los clientes. En caso de necesitar dar persistencia a la información, el servidor se comunicará con la base de datos.

Por ello, en el servidor primero se debe establecer una conexión a modo de cliente de la base de datos. Para ello, se ha empleado un driver de GO *go.mongodb.org/mongo − driver* (*Driver de MongoDB para GO*, s.f.). Este permite conectarse de manera sencilla con bases de datos MongoDB, solamente se necesita una cadena de conexión que nos proporciona la base de datos.

La inserción de la cadena en el código se puede efectuar o bien por variables de entorno o en caso de que no exista dicha variable, se le solicita al usuario por consola. Además de esta cadena se requiere del uso del paquete *context* (*Paquete Context de GO*, s.f.) de la librería estándar de GO.

Context, como indica su nombre, permite dotar de un contexto a funciones o métodos, es decir, con este paquete se puede definir un tiempo máximo para esperar por una llamada a un método bloqueante. Un método bloqueante es aquel que depende de un evento externo, por ejemplo en este caso sería una llamada a la base de datos.

Al utilizarse *context* se evitan casos en los que la base de datos no esté disponible y el código se quede esperando de manera infinita. En todas las llamadas a la base de datos que se realicen, siempre se establecerá un contexto previamente.

Una vez establecida la conexión con la base de datos, se asigna la misma a una instancia, que se empleara en todas las llamadas que se efectúen a la base de datos, para cualquier tipo de operación que se quiera realizar. La conexión con la base de datos usando *context* se puede ver en el código 5.1.

Código 5.1: Ejemplo establecimiento de conexión a la base de datos

```
func ConnectDB() {
    cadena := os.Getenv("CADENA_CONEXION")
    if cadena == "" {
        fmt.Println("Inserta la cadena de conexion de la base de datos")
        fmt.Scanf("%v\n", &cadena)
    }
    client, err := mongo.NewClient(options.Client().ApplyURI(cadena))
    if err != nil {
        log.Fatal(err)
    }
    //Establecemos un context antes de conectarnos
    ctx, _ := context.WithTimeout(context.Background(), 10*time.Second)
    err = client.Connect(ctx)
    if err != nil {
        log.Fatal(err)
    }
    err = client.Ping(ctx, readpref.Primary())
    if err != nil {
        log.Fatal(err)
    }
    InstanceDB = MongoConection{
        DB: client.Database(nombreDB),
        Client: client,
    }
    //Creo los indices
    CreateIndexUniqueUsers()
    CreateIndexComposeUserProyectList()
    CreateIndexListIDinTask()
}
```

También hay que destacar que al establecer la conexión, seguidamente se ejecuta la creación de los índices mencionados en el apartado 5.3.2. En el código 5.2 se puede ver como se crea un índice para la colección de usuarios.

Código 5.2: Ejemplo creación índice

```
func CreateIndexUniqueUsers() {
    coleccion := InstanceDB.DB.Collection("users")
    _, err := coleccion.Indexes().CreateOne(
        context.Background(),
        mongo.IndexModel{
            Keys: bson.D{{Key: "email", Value: 1}},
            Options: options.Index().SetUnique(true),
        },
    )
    if err != nil {
        fmt.Println(err)
    }
}
```

Lo siguiente consiste en inicializar el enrutador con el cual se definen las rutas y métodos HTTP deseados. Para disponer de un enrutador se hace uso de la librería *github.com/gorilla/mux*

(*Paquete gorilla/mux*, s.f.). Esta permite definir grupos de rutas (usuarios, proyectos, listas, relaciones y tareas), sobre las que los clientes de la API pueden hacer peticiones.

En cada uno de estos grupos se definen las rutas concretas para una colección de la base de datos. En el código 5.3 se puede ver la definición de las distintas rutas de los usuarios.

Código 5.3: Ejemplo rutas usuario de la API

```
func Task(r *mux.Router) {
    //Creo una tarea
    r.HandleFunc("/task", handlers.CreateTask).Methods("POST")
    //Recupero una tarea por su ID
    r.HandleFunc("/tasks/{taskID}", handlers.GetTaskByID).Methods("GET")
    //Recupero las tareas pertenecientes a una lista dado el ID de la lista
    r.HandleFunc("/tasks/list/{listID}", handlers.GetTasksByList).Methods("GET")
    //Recupero las tareas pertenecientes a una lista dado el ID de la lista
    r.HandleFunc("/tasks/list/{listID}", handlers.DeleteTasksByList).Methods("DELETE")
    //Actualizo una tarea
    r.HandleFunc("/tasks/{id}", handlers.UpdateTask).Methods("PUT")
    //Borro una tarea
    r.HandleFunc("/tasks/{id}", handlers.DeleteTask).Methods("DELETE")
}
```

Cada una de las rutas definidas dispone de lo que se conoce como un handler (manejador). Este se encarga de gestionar la petición que se realiza sobre la ruta a la que está asociado. Los manejadores son simples funciones que reciben dos parámetros: la petición enviada por el cliente y la respuesta que se le enviara.

En la función que representa el handler, se puede acceder tanto a los parámetros puestos en la propia URL de la petición como a los del cuerpo de esta, como se puede observar en el código 5.4.

Código 5.4: Ejemplo handler de una ruta de la API del servidor

```
func UpdateUser(w http.ResponseWriter, r *http.Request) {
    w = utils.SetRefreshToken(w, r)
    var user models.User
    json.NewDecoder(r.Body).Decode(&user)
    params := mux.Vars(r)
    email := params["email"]
    resultado := models.UpdateUser(email, user.Status)
    if !resultado {
        w.WriteHeader(400)
        w.Write([]byte("No se actualizo el usuario"))
    } else {
        w.Write([]byte("Se actualizo el usuario"))
    }
}
```

Una vez se recuperan los datos necesarios de la petición, se realizan las llamadas a los modelos. Estos representan la capa encargada de comunicarse con la base de datos. Desde los modelos se efectúan todas las operaciones (inserción, recuperación, actualización y borrado) de los documentos de la base de datos en la colección pertinente. Para cada una de estas operaciones se utilizan métodos definidos por el driver de MongoDB con el que establecemos la conexión. Un ejemplo de como se recuperan datos de la base de datos se puede ver en el

código 5.5.

Código 5.5: Ejemplo recuperación datos de la base de datos

```
func GetTaskByID(taskIDstring string) Task {
    //Creo un contexto
    ctx, cancel := context.WithTimeout(context.Background(), 10*time.Second)
    defer cancel()
    //Obtengo la coleccion
    coleccion := config.InstanceDB.DB.Collection("tasks")
    taskID, _ := primitive.ObjectIDFromHex(taskIDstring)
    filter := bson.M{"_id": taskID}
    var task Task
    err := coleccion.FindOne(ctx, filter).Decode(&task)
    if err != nil {
        log.Println(err)
    }
    return task
}
```

Entre las funciones usadas para interaccionar con la base de datos, destacan de cara a la actualización y borrado de un documento, el uso de *FindOneAndDelete*, *FindOneAndUpdate* y *FindOneAndReplace*. Estas permiten evitar que se efectúen simultáneamente otras operaciones sobre el mismo documento que están afectando las operaciones mencionadas, logrando así evitar problemas de concurrencia.

Retomando el tema de los handlers, previamente a que una petición a una ruta llegue a su manejador correspondiente, existen una serie de funciones intermedias denominadas middlewares.

Una función middleware intercepta todas las peticiones antes de que las procese el handler correspondiente. En estas funciones middleware se pueden hacer comprobaciones y producir cambios tanto en la petición realizada, como incluso en la respuesta que después se devolverá al cliente.

En el proyecto, se han establecido tres middlewares. Dos de ellos son referentes a aspectos de seguridad y, por lo tanto, se comentarán posteriormente en el apartado 5.6. El restante se encarga de modificar la cabecera de todas las respuestas para indicar que el contenido de la respuesta está en formato JSON.

5.5.2 Cliente

Por otra parte, se ha desarrollado un software cliente. Aunque desde el punto de vista de los usuarios, este funciona como un único elemento, aunque realmente el mismo se encuentra dividido en dos partes. Una parte visual con la que interactúan los usuarios, el *frontend*. Y el *backend*, donde se realiza realmente toda la funcionalidad ofrecida en el *frontend*.

Esta dualidad, en la que, por un lado, se recogen los datos en la parte del *frontend*, para finalmente procesarlos en el *backend*, se lleva a cabo utilizando la librería *lorca*, utilizada para desarrollar la interfaz de usuario.

En el cliente, inicialmente desde el *backend*, se inicializa la interfaz de usuario con la primera vista que visualizaran los usuarios. En la inicialización se pueden establecer parámetros como el ancho y alto de la ventana a mostrar, entre otros. La inicialización se puede ver en el código 5.6. En el caso del proyecto, la primera vista que se carga es la pantalla de inicio de sesión.

Código 5.6: Ejemplo carga del interfaz de usuario usando Lorca

```
func InitUI() {
    //Inicializo
    UI, _ = lorca.New("", "", 1250, 800, "--allow-insecure-localhost")
    //Cargo la primera vista
    ChangeView(config.PreView + "user/login.html")
}
```

La librería no solo permite cargar ficheros HTML para que los visualicen los usuarios. También permite establecer una comunicación desde la parte del *frontend* donde se usa JS, con la parte del *backend* donde se usa GO. Permitiendo que desde funciones JS se pueden realizar llamadas a funciones GO. De esta manera, el código JS únicamente se emplea para dotar de funcionalidad a los elementos HTML con los que interactúan los usuarios. Y es en el *backend* con GO donde se procesan todas las interacciones y peticiones de los usuarios.

El proceso por el cual se puede realizar esta comunicación se denomina *bind* y se puede ver como se realiza en el código 5.7.

Código 5.7: Ejemplo uso del metodo bind de Lorca

```
UI.Bind("changeToRegister", func() {
    ChangeView(config.PreView + "user/register.html")
})
```

El proceso de uso del método *bind* en las dos partes (*frontend* y *backend*) se resume en:

1. En GO se declaran funciones usando el método *bind*.
2. Desde JS empleando una promesa se llama a una función declarada en GO con *bind*.
3. Al terminar la función GO, devuelve los datos pertinentes a JS.
4. Los datos llegan a JS, como un parámetro dentro de la función asociada al cumplimiento de la promesa.
5. Desde JS se informa al usuario.

El uso de promesas en JS, se debe a que la función en GO puede tardar un tiempo en procesarse, por ello se usan promesas para evitar así bloquear el código JS. Una vez en GO se haya procesado la función, se devolverán los resultados. La llamada a una función en GO desde JS se puede ver en el código 5.8.

Código 5.8: Ejemplo llamada desde JS a código GO

```
function changeToProyectConfig(proyectID){
```

```
    changeToProyectConfigGO(proyectID).then(
        function(resultados){
            let tokenExpire = resultados[1];
            let changeOK = resultados[0];
            if (tokenExpire){
                alert('El Token de la sesión expiro')
                exitSesion();
            }else{
                if (changeOK == false){
                    alert('Ya no esta disponible el proyecto')
                    changeHome(true)
                }
            }
        }
    )
}
```

Sobre la interfaz de usuario que se ha desarrollado, no se ha optado por disponer de una única página e ir modificándola dinámicamente, lo que se conoce como aplicaciones SPA (Single-page application). Se ha decidido desarrollar distintas interfaces para cada una de las acciones o elementos del sistema (añadir un proyecto, cambiar la configuración de una lista, etc.).

Para la carga de las distintas interfaces, se ha seguido con la metodología explicada anteriormente, donde el cliente solo necesita pulsar un botón HTML. Dicho botón dispone de un evento asociado a una función JS, desde la que se realiza la comunicación con la función de GO correspondiente.

Desde la función del *backend*, se realizará la carga combinando la ya comentada librería *lorca*, junto a la librería *html/template*. Esta última permite cargar plantillas HTML.

La carga de ficheros HTML con *html/template* puede verse como se efectúa en el código 5.9. Esta sigue una serie de pasos:

1. Se carga el fichero HTML que se quiera mostrar al usuario.
2. Se crea una estructura de GO (*struct*), en la cual se pueden añadir todo tipo de estructura de datos de GO, incluidas funciones.
3. Se ejecuta la plantilla cargada junto a los datos de la estructura.

Código 5.9: Ejemplo carga y ejecución de una plantilla html/template con datos

```
func ChangeViewWithValues(nombreVista string, emails []string) {
    tmpl, err := template.ParseFiles(nombreVista)
    if err != nil {
        log.Fatal(err)
    }
    dataStruct := Data{
        User: models.UserSesion,
        Datos: models.DatosUsuario,
        Emails: emails,
    }
    buff := bytes.Buffer{}
```

```
12    tmpl.Execute(&buff, dataStruct)
13    loadableContents := "data:text/html," + url.PathEscape(buff.String())
14    UI.Load(loadableContents)
15}
```

Para que los datos de la estructura declarada, acaben pasándose al HTML en los lugares que el desarrollador desee que ocupen, es necesario mencionarlos en el fichero HTML previamente. Para referenciar los datos de la estructura, se deben escribir entre llaves dobles ({{}}), como se puede ver en el código 5.10.

Dentro de estos caracteres, empleando el símbolo punto (.), se puede acceder a los distintos elementos de la estructura. En caso de tratarse de por ejemplo otro *struct*, se utilizará otro símbolo punto para acceder a los elementos de este y así sucesivamente hasta llegar a unidades básicas como enteros o cadenas de texto.

La librería además permite realizar iteraciones sobre los datos, evaluar expresiones condicionales y demás. Una vez se ha ejecutado la plantilla con los datos pasados, usando *lorca* se cargará en la interfaz visual la plantilla para mostrársela al usuario.

Código 5.10: Ejemplo mostrar datos en una plantilla html/template

```
1<select class="form-control" name="users" id="users">
2    {{range .Emails}}
3        <option value={{.User}}>{{.User}}</option>
4    {{end}}
5</select>
```

Desde el *frontend*, no solo se vinculan a los botones del HTML funciones para cargar nuevas interfaces. También se permiten a los usuarios hacer todo tipo de operaciones CRUD sobre los tres elementos del sistema (proyectos, listas y tareas).

En estos casos, la función en GO llamada desde JS, se encargará de enviar una petición a la API del servidor. Una vez el servidor devuelva una respuesta, esta se pasará a la función de JS, para que este último informe al usuario mediante una alerta.

Por la naturaleza de la aplicación, tras una acción exitosa se ejecutará una función para volver a cargar la página en la que está el usuario. En caso de haber modificaciones en los datos que se estuvieran visualizando, dicha pantalla variará de la anterior.

Además de las funcionalidades comentadas y exceptuando a todas aquellas relacionadas con la seguridad, el cliente también realiza otras funcionalidades. Dos de ellas, las más destacables, son referentes a las actualizaciones de los elementos del sistema y a realizar cada cierto tiempo una comprobación para notificar al usuario si se han producido cambios sobre los elementos relacionados con este.

Al efectuar actualizaciones sobre proyectos, listas y tareas, surge un problema de concurrencia. Este tipo de problema se da cuando dos usuarios tratan de modificar simultáneamente la misma información. Para evitar este problema y aprovechando el estudio efectuado sobre técnicas criptográficas, se ha optado por utilizar las funciones *hash* para solventar el proble-

ma de la forma que describo a continuación:

En las estructuras de datos (proyectos, listas y tareas), se crea un campo *Check* en el documento de la base de datos que las representa. Dicho campo almacena un *hash* de todos los datos de esa estructura. Por ejemplo, para una tarea se generará un *hash* de los diferentes campos de la tarea (fecha, estado, usuarios adjuntos, etc.).

Cuando se ejecuta una operación de actualización, se envía la nueva estructura con los campos actualizados cifrados y además dos campos. El nuevo campo *Check*, que contiene el nuevo *hash* de los datos y un campo *UpdateCheck*. En dicho campo se adjunta el valor del *hash* anterior a modificar los cambios.

Al llegar al servidor se hace una comprobación sobre el campo *Check* de la tarea que hay en la base de datos, con el campo *UpdateCheck* de la nueva tarea. En caso de coincidir, se puede afirmar que la tarea no ha sido modificada. En caso contrario, se informa al usuario, que otro usuario ya ha efectuado una actualización y existe un conflicto.

Con el proceso descrito, se logra que el servidor sea capaz de gestionar la concurrencia sin necesidad de conocer los datos del sistema.

Por otro lado, se encuentra la funcionalidad que permite alertar al usuario que se han producido cambios en alguno de los proyectos, listas o tareas relacionados con este. Para ello, en el cliente, al descifrar cualquier de estos elementos, se guarda una copia en local, de lo que esté visualizando el cliente.

Desde el *frontend* si el cliente no cambia de pantalla, al pasar un tiempo se llama a una función en GO. Dicha función realiza una petición al servidor para traer la información que esté visualizando el usuario, ya sean los proyectos con sus listas o las tareas de una lista. Una vez recibidos se realizan en orden las siguientes comprobaciones:

1. Se comprueba que la cantidad de elementos en local, frente a los recién recuperados, es la misma.

2. En caso de que lo anterior se cumpla, se revisara la información de cada uno de los elementos recuperados frente a la que tienen los elementos almacenados en el cliente, para observar alguna modificación. Si lo anterior no se cumple, este paso directamente no se efectúa.

En caso de llegar al segundo punto y que no se observe ningún cambio, no se le comunicara nada al usuario. Por el contrario, se le indicará al usuario si desea actualizar la página, dado que se han realizado cambios sobre los elementos que está visualizando.

El tiempo de comprobación de las actualizaciones es configurable desde el cliente.

5.6 Capa de seguridad

La parte fundamental del desarrollo del proyecto es la referente a dotar al mismo de una gruesa capa de seguridad, no solo en la confidencialidad de la información, sino también en otros aspectos como la integridad, autenticación y el no repudio.

En las siguientes secciones se detallarán los algoritmos, técnicas y procesos escogidos e implementados para lograr dotar de una robusta seguridad al sistema. La notación utilizada es la siguiente:

- $h_f(m)$: $Hash$ resultante de aplicar una función $hash$ f a un texto en claro m.
- K_{pub_u} : Clave pública generada por el algoritmo RSA para un usuario u.
- K_{priv_u} : Clave privada generada por el algoritmo RSA para un usuario u.
- $E_k^a(m)$: Cifrar el texto en claro m con el algoritmo de cifrado a usando la clave k.
- $D_k^a(c)$: Descifrar el criptograma c con el algoritmo de cifrado a empleando la clave k.
- $K_{servidor_u}$: Clave empleada por un usuario u para autenticarse en el servidor.
- K_{BD_u} : Clave de autenticación en el servidor del usuario u derivada mediante la función $Bcrypt$ que se encuentra almacenada en la base de datos.
- VI : Vector de inicialización.
- C_u : Certificado del usuario u.
- r : Firma digital.
- M_{AC} : Mensaje introducido al arrancar el servidor del cual se obtienen la K_{AC} y el VI.
- K_u : Clave utilizada para cifrar con AES para el usuario u.
- K_{item_e} : Clave aleatoria utilizada para cifrar/descifrar la estructura e.
- $emailPassword_u$: La concatenación de las cadenas que representan la dirección de correo electrónico del usuario u y su contraseña.
- $B(k, n)$: $Hash$ obtenido de aplicar la función $Bcrypt$ con una serie de rondas de sal n, sobre una clave k.
- $sliceRND$: Un *slice* (véase McGranaghan y Bendersky, s.f.) de bytes aleatorios.

5.6.1 Autoridad Certificadora

El primer aspecto sobre la seguridad del sistema, recae del lado del servidor, en el cual se realiza el despliegue de una autoridad certificadora (AC). Dicha AC, será la encargada de emitir certificados para los usuarios del propio sistema. Como se mencionó anteriormente, la AC se podría trasladar a un servidor diferente en un entorno en producción.

Para el despliegue de la AC, es necesario crear para la misma un certificado junto a un par de claves RSA pública y privada. El certificado generado sigue el formato x.509.

Con respecto a la creación del certificado, lo más destacable es que a la hora de generar el mismo, se debe indicar que este certificado es de una autoridad certificadora, esto se realiza indicándolo en uno de los campos del propio certificado.

Junto al certificado C_{AC} de la AC, también se generan un par de claves RSA-4096. La clave privada $K_{priv_{AC}}$ que se utilizará para firmar los certificados que emitirá la misma y la clave publica $K_{pub_{AC}}$ correspondiente, que se adjunta en el C_{AC}.

Tanto el C_{AC}, como la $K_{pub_{AC}}$ se almacenan ambos en ficheros con formato PEM. Este fue formalizado en la RFC 7468 (Josefsson y Leonard, 2015). En cambio, para garantizar la confidencialidad de la $K_{priv_{AC}}$, esta se almacena cifrada.

Para el cifrado se emplea una clave K_{AC} y un VI. Estos dos valores son generados a partir de una frase M_{AC} introducida por el responsable del servidor. Dicha frase puede introducirse por variables de entorno (no recomendable para entornos en producción) o, por el contrario, hacerlo por consola cuando el servidor nos la solicite al no detectar la variable de entorno.

Sea por el método de inserción que sea, una vez el servidor dispone de M_{AC}, procede a generar la K_{AC} y el VI, para ello aplica una función $hash$ sobre M_{AC} de la siguiente manera: $h_{sha512}(M_{AC})$.

Una vez se dispone del $hash$ se utilizan los primeros 16 bytes para el VI y los siguientes 32 bytes para la K_{AC}. Con estos dos valores se realiza el cifrado, por lo que se almacena en un fichero $E^{AES256}_{K_{AC}}(K_{priv_{AC}})$.

5.6.2 Autenticación

Para la utilización del sistema es necesario obtener una cuenta en el mismo. Para generar una cuenta debe llevarse a cabo un proceso de registro. Dicho proceso se efectúa tanto en el servidor como en el cliente.

Comenzando el proceso en el cliente, se le solicita al usuario una dirección de correo electrónico (email) junto a una contraseña. Para la contraseña se han especificado ciertos requisitos:

- Debe contener de 8 a 15 caracteres.
- Mínimo una letra minúscula.
- Mínimo una letra mayúscula.
- Mínimo un dígito.
- Mínimo un símbolo de entre los siguientes: @,$,!,%,*,?,&.

Una vez se valida la contraseña, se procede, a partir de estos dos campos, a obtener una serie de valores. Por ejemplo, para el caso del registro de un usuario u_1, lo primero que se realiza es aplicar $h_{sha512}(emailPassword_{u_1})$. Con esto se obtiene un *hash* de un total de 64 bytes, sobre los que se hace un reparto en diferentes variables tal que así:

- Los primeros 16 bytes del *hash* se asignan para la $K_{servidor_{u_1}}$.
- Los siguientes 16 bytes para el VI.
- Los 32 bytes restantes son utilizados para la K_{u_1}.

Una vez obtenidos estos tres valores, se procede con la generación de un par de claves RSA-2048 para el usuario. Se crean tanto la $K_{pub_{u_1}}$, como la $K_{priv_{u_1}}$. Esta última debe mantenerse en secreto, por ello es cifrada con el VI y la K_{u_1} generados anteriormente.

El proceso de cifrado de la $K_{priv_{u_1}}$, es el siguiente: $E^{AES256}_{K_{u_1}}(K_{priv_{u_1}})$. Una vez realizados estos cálculos se hace una petición de registro al servidor enviando:

- El email del usuario u_1.
- La clave del servidor $K_{servidor_{u_1}}$.
- La clave pública del usuario $K_{pub_{u_1}}$.
- Y la clave privada cifrada $E^{AES256}_{K_{u_1}}(K_{priv_{u_1}})$.

La K_{u_1}, empleada para cifrar la $K_{priv_{u_1}}$ no se envía al servidor, en ningún momento. Una vez llegan los datos al servidor, se comprueba que el email del usuario no exista ya en el sistema. Si este ya existe, se le devuelve un error al cliente.

En caso de no existir el email, antes de almacenar los datos enviados en la base de datos, se realiza un proceso de derivación de clave sobre la $K_{servidor_{u_1}}$. El proceso se efectúa utilizando la función *Bcrypt* de la librería *crpyto* de GO.

Esta función garantiza un almacenamiento seguro de la clave. Al aplicar la función se obtiene $K_{BD_{u_1}} = B(K_{servidor_{u_1}}, 12)$, dicho valor $K_{BD_{u_1}}$ es almacenado en la base de datos. De esta manera se protege la clave con la que después se validara el inicio de sesión de un usuario. El resto de los valores enviados, se almacenan tal cual llegaron al servidor.

Los datos son almacenados en la base de datos como campos de un documento que forma parte de la colección *users* de la base de datos. Antes de mandar la respuesta de que el proceso ha sido exitoso al cliente, se lleva a cabo otro proceso en el servidor.

La autoridad certificadora se encarga de emitir un certificado digital para el usuario que se está registrado. Por lo que la generación del certificado es la siguiente: $C_{u_1} = E^{RSA}_{K_{priv_{AC}}}(K_{pub_{u_1}})$. Una vez generado, es almacenado en formato PEM en la ruta */certs/users* del servidor.

Finalmente, el servidor envía la respuesta al cliente notificándole que el registro se ha ejecutado con éxito. Desde la interfaz visual del cliente, se notificará al usuario de que su cuenta

ya está disponible en el sistema. Para hacer uso de esta, se debe realizar un proceso de inicio de sesión.

El proceso de inicio de sesión solicita al igual que en el registro, la dirección de correo electrónico del usuario u_1 junto a su contraseña. Nuevamente se obtiene un *hash* de la misma manera: $h_{sha512}(emailPassword_{u_1})$.

Este *hash* se divide igual que en el registro en las tres variables mencionadas anteriormente: la $K_{servidor_{u_1}}$, el VI y la K_{u_1}. Pese a obtener estos tres valores, solamente se envían el *email* del usuario y la $K_{servidor_{u_1}}$ en la petición de inicio de sesión al servidor.

Al recibir estos valores en el servidor, se comprueba que el email enviado, pertenece a un usuario almacenado en la base de datos. En caso de que el usuario no exista, el servidor devuelve un error. De lo contrario, se calcula $B(K_{servidor_{u_1}}, 12)$ y se compara con la $K_{BD_{u_1}}$ almacenada en la base de datos. Si son iguales, se recuperaran la $K_{pub_{u_1}}$ y $E^{AES256}_{K_{u_1}}(K_{priv_{u_1}})$).

Estas dos claves son enviadas al cliente, donde se le notifica al usuario que el proceso ha resultado exitoso. En caso de que al comparar los valores anteriores, sean distintos, se le notificara al cliente de un error.

5.6.3 Cifrado

Previamente a detallar a nivel matemático los procesos de cifrado/descifrado de la información e intercambio de claves, se comentará desde un nivel alto de abstracción como se trabaja con la información del sistema tanto en el cliente como en el servidor.

El sistema se ha desarrollado buscando garantizar en todo momento la máxima privacidad y confidencialidad de la información con la que se trabaja, esto ha dado lugar a que el intercambio de información se produzca de la siguiente manera:

En el cliente se aplican técnicas y algoritmos para garantizar la confidencialidad de la información que se ha considerado sensible. Para el envío al servidor se ha usado HTTPS para garantizar aún más la seguridad, dotando a la misma de una doble capa de cifrado. La información una vez llega al servidor ya se encuentra totalmente cifrada. Este simplemente se encarga de almacenarla.

Cuando un cliente solicite información al servidor, este la envía cifrada tal cual la envió el cliente previamente, es decir, no se efectúa ningún tipo de proceso de descifrado en el servidor. Esto consigue que el conocimiento en el lado del servidor sobre la información sea nulo. El envío desde el servidor al cliente también utiliza HTTPS.

Será finalmente en el cliente, donde se termine por descifrar la información y mostrársela al usuario. Todo este proceso puede verse de forma gráfica en la figura 5.4.

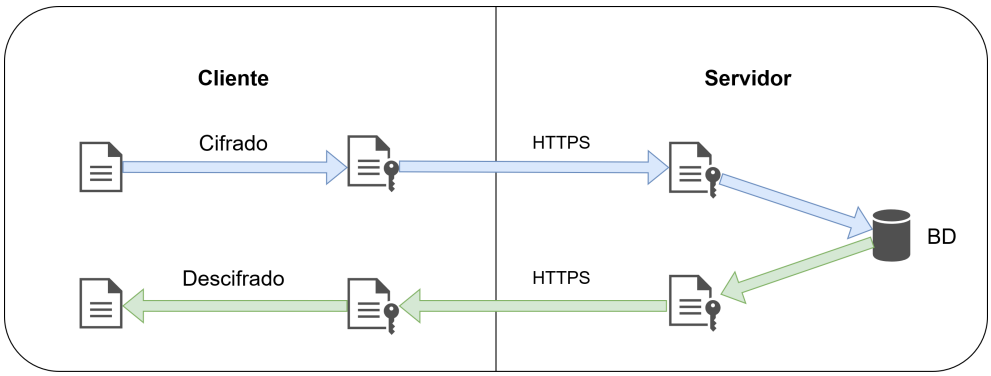

Figura 5.4: Estado de la información en las peticiones y respuestas entre cliente y servidor

Para trasladar esto al código desarrollado, se utilizan dos *structs* en el cliente. Una de estas permite almacenar, por ejemplo, un proyecto sin cifrar tal cual con todos los datos de este. La otra estructura, cuenta con un campo *CipherData*, en el que se almacena cifrada toda la información relevante y sensible del proyecto. El *struct* con el campo *CipherData*, será el usado en el envío al servidor.

De cara a la persistencia de la información en la base de datos, la información se almacena tal cual se puede ver en la figura 5.5. De esta manera, en caso de que la base de datos se viera comprometida, la información sensible y relevante se encuentra totalmente protegida de un atacante.

Figura 5.5: Documento de la colección de tareas de la base de datos

Por último, añadir que por motivos de dotar de la máxima privacidad a la información del sistema, se ha realizado una división interna de las tres estructuras de datos (proyectos, listas y tareas), en lo referente a las claves utilizadas para los procesos de cifrado y descifrado de estas.

La división consiste en emplear una clave única para cifrar/descifrar cada proyecto. Y dentro de este proyecto, se usará una clave distinta para cifrar/descifrar tanto la propia lista como las tareas de esa lista.

A continuación, se va a detallar a nivel matemático el proceso de cifrado que se hace en el sistema. Tomando como referencia el cifrado de un nuevo proyecto. En primer lugar, en la parte *frontend* del cliente son recogidos los datos del nuevo proyecto y trasladados al *backend* para dotarlos de la mayor seguridad posible antes de enviarlos al servidor y ser almacenados.

Una vez recibidos los datos del proyecto p_1 en el *backend*, se procede a efectuar un cifrado mediante el algoritmo simétrico AES con modo de operación CTR, de los campos que se han considerado sensibles sobre este.

Para el cifrado, es necesario previamente generar una $K_{item\_p_1}$ y un VI. Para la generación de estos valores, primero se obtiene un $sliceRND$, al cual se le aplica una función *hash* tal que $h_{sha512}(sliceRND)$. Ahora se efectúa un reparto de los bytes del *hash* tal que:

- Los primeros 16 bytes se asignan al VI.
- Los siguientes 32 bytes a $K_{item\_p_1}$.

La generación del $sliceRND$, se realiza con el paquete *rand* de la librería *crypto* de GO (Libreria Crypto, s.f.). Este paquete proporciona un generador de números aleatorios criptográficamente seguro.

Obtenidos estos dos valores, se procede a cifrar los campos sensibles de p_1, tal que: $E^{AES256}_{K_{item\_p_1}}(p_1)$. Dicho resultado se almacena como se comentó anteriormente en el campo $CipherData$ de un *struct* que se envía al servidor.

Una vez enviado, es necesario almacenar de forma segura para cada uno de los usuarios asignados al proyecto, la clave $K_{item\_p_1}$ utilizada en el cifrado y que se utilizara posteriormente para descifrar.

La clave $K_{item\_p_1}$, es cifrada con la clave pública de cada uno de los usuarios asignados, es decir, ahora no se usará un cifrador simétrico, sino uno asimétrico como RSA.

El proceso comienza con la recuperación de las claves públicas de cada uno de los usuarios asignados (u_1, u_2, ..., u_n), donde n es el número de usuarios asignados. Para cada uno de estos se realiza un cifrado:

$$
\begin{gathered}
E^{RSA}_{K_{pubu_1}}(K_{item\_p_1}) \\
E^{RSA}_{K_{pubu_2}}(K_{item\_p_1}) \\
\vdots \\
E^{RSA}_{K_{pubu_n}}(K_{item\_p_1})
\end{gathered}
$$

Las claves una vez cifradas son enviadas al servidor para almacenarlas en la colección de relaciones.

De cara a las listas del proyecto, el proceso es el mismo. Teniendo en cuenta que si estás, por ejemplo, pertenecen a p_1 todas ellas emplean una clave de cifrado/descifrado distinta de

la $K_{item\_p_1}$ utilizada por el proyecto.

Para una lista l_1 perteneciente al proyecto p_1, su clave aleatoria $K_{item\_l_1}$, será cifrada como se ha comentado anteriormente. Teniendo en cuenta que solamente se cifrara para los usuarios asignados a la propia lista y no para todos los usuarios del proyecto.

Las claves cifradas de las listas pertenecientes a un proyecto, se almacenan en un array del documento que representa la relación usuario-proyecto de la colección *relations*, se puede ver en la figura 5.6.

Figura 5.6: Documento de la colección de relaciones de la base de datos

Por último, sobre las tareas de una lista, añadir que estás en caso de pertenecer a la lista l_1, usan la misma clave $K_{item\_l_1}$ que la que se utilizó para cifrar la lista. Junto a un VI diferente para cada una de las tareas.

Por otra parte, se encuentra el proceso de descifrado, este es algo más complejo que el cifrado. Retomando el ejemplo anterior, donde se ha cifrado un proyecto p_1, si el usuario u_1 perteneciente al proyecto p_1 quiere descifrarlo, debe realizar una serie de pasos:

Primero debe recuperarse la clave aleatoria empleada en el cifrado de la información del proyecto, que se encuentra cifrada con la clave pública del usuario $E^{RSA}_{K_{pub_{u_1}}}(K_{item\_p_1})$.

Para descifrar dicha clave es necesario emplear la clave privada del usuario $K_{priv_{u_1}}$. Esta clave actualmente se encuentra cifrada con la K_{u_1}, por lo tanto, se deben realizar los siguientes procesos de descifrado:

1. Descifrar $K_{priv_{u_1}}$ con la K_{u_1}, tal que $K_{priv_{u_1}} = D^{AES256}_{K_{u_1}}[E^{AES256}_{K_{u_1}}(K_{priv_{u_1}})]$.
2. Con la $K_{priv_{u_1}}$ obtener $K_{item\_p_1} = D^{RSA}_{K_{priv_{u_1}}}[E^{RSA}_{K_{pub_{u_1}}}(K_{item\_p_1})]$.
3. Finalmente descifrar el proyecto $p_1 = D^{AES256}_{K_{item\_p_1}}[E^{AES256}_{K_{item\_p_1}}(p_1)]$.

De cara al descifrado de una lista o una tarea, el proceso es el mismo. Pero como se comentó anteriormente, en este caso se solicitaría la clave empleada para cifrar la lista. Dicha clave también se encuentra cifrada con la clave pública del usuario: $E^{RSA}_{K_{pub_{u_1}}}(K_{item\_l_1})$.

Por último, dado que se ha escogido el modo de operación CTR para cifrar con AES. A la hora de realizar actualizaciones en las estructuras de datos, al cifrar la nueva estructura resultante, se ha optado por utilizar un VI distinto al usado anteriormente, para garantizar

al máximo la seguridad de este modo de operación.

Esto se debe a que en CTR la reutilización del vector de inicialización, provoca la reutilización del flujo de bits que se emplea para combinar mediante una XOR con el texto en claro. Esto conllevaría una brecha de seguridad, por lo tanto, se ha optado por generar un VI cada vez que se efectúa un cifrado en el sistema.

Por último, añadir que los VI que se emplean al cifrar y que después son necesarios para descifrar la información, son almacenados en los primeros 16 bytes del *slice* de bytes que representa el criptograma resultante del proceso de cifrado.

5.6.4 Firma digital

La incorporación de la firma digital en el proyecto se ha dado en dos partes diferenciadas. Por un lado, para firmar los elementos adjuntos a una tarea (documentos y enlaces). En este caso se obtiene un *hash* de todos los datos del documento o de la dirección (URL) del enlace, tal que $h_{sha256}(documento/enlace)$. Dicho *hash* será firmado por el usuario que adjunta el documento/enlace.

La otra parte en la que se ha incorporado, ha sido para dotar de mayor validez y no repudio al registro de eventos establecido para las tareas del sistema. Dicho registro sirve para que los usuarios puedan registrar los eventos vinculados a una tarea. Los eventos y los usuarios que pueden firmar estos, son los siguientes:

- La creación de una tarea, solo puede ser firmada por el creador de esta.
- La recepción de una tarea, puede ser firmada por cualquier usuario asignado a esta.
- La finalización de una tarea, únicamente puede ser firmada cuando se haya finalizado esta y por usuarios asignados a la misma.

Para firmar estos eventos, se obtiene un *hash* de una cadena (*string*) que se ha definido previamente. En dicha cadena se indican una serie de valores:

- El usuario que efectúa la firma.
- El evento que firma (creación, recepción, finalización).
- El identificador único de la tarea.
- La fecha de la firma.

Por ejemplo, el proceso de firma para que un usuario u_1, pueda firmar un evento es el siguiente:

Primero, se obtiene el *hash* correspondiente tal que $h_{sha256}(evento)$. Una vez realizado el *hash*, se procede a firmarlo, para ello es necesario utilizar la clave privada del usuario $K_{priv_{u_1}}$. Esta se encuentra cifrada en memoria en el cliente, por ello es necesario descifrarla para poder

usarla.

El descifrado de esta, se hace empleando la K_{u_1}, como se indicó anteriormente: $K_{priv_{u_1}} = D^{AES256}_{K_{u_1}}[E^{AES256}_{K_{u_1}}(K_{priv_{u_1}})]$.

Una vez se dispone de esta, se procede a efectuar el cifrado (firma) del *hash*, obteniendo así $r = E^{RSA}_{K_{priv_{u_1}}}(h_{sha256}(evento))$. Dicho valor, r, es la firma digital. Cada firma es almacenada en el campo correspondiente del *struct* que representa a la tarea. Junto a la firma se almacena el *hash* sobre el que se ha efectuado la firma.

En el caso de que un usuario u_2, quiera validar la firma digital r que se ha realizado anteriormente, debe realizar una serie de pasos:

1. Solicitar al servidor la $K_{pub_{u_1}}$.
2. Con esta se descifra la firma digital: $D^{RSA}_{K_{pub_{u_1}}}(r)$. Esto último, es equivalente a efectuar: $D^{RSA}_{K_{pub_{u_1}}}[E^{RSA}_{K_{priv_{u_1}}}(h_{sha256}(evento))]$.
3. Finalmente compara $h_{sha256}(evento)$ con $D^{RSA}_{K_{pub_{u_1}}}[E^{RSA}_{K_{priv_{u_1}}}(h_{sha256}(evento))]$.
4. En caso de coincidir, se le indica al usuario que la firma es válida. De lo contrario, se le notifica del error.

5.6.5 Certificados digitales

Para dotar de mayor validez a la validación de la firma, cuando el usuario u_2 solicita la $K_{pub_{u_1}}$ antes de validar la firma, también solicita a la autoridad certificadora el certificado C_{u_1} junto a la $K_{pub_{AC}}$.

Con estos dos valores se valida que la $K_{pub_{u_1}}$ es igual a $D^{RSA}_{K_{pub_{AC}}}[E^{RSA}_{K_{priv_{AC}}}(K_{pub_{u_1}})]$. En caso de coincidir ambos valores, se puede saber con seguridad que la $K_{pub_{u_1}}$ es realmente del usuario u_1. Una vez comprobado esto, se lleva a cabo la validación de la firma digital.

5.6.6 Roles

Se ha incorporado un sencillo sistema de roles sobre las dos principales estructuras de datos (proyectos y listas), para controlar las acciones o permisos, que pueden efectuar los distintos usuarios asignados a estas estructuras.

Tanto para los proyectos como para las listas, se han establecido dos tipos de roles o perfiles de usuario. Por un lado, los administradores que dispondrán de todas las capacidades que permite el sistema sobre las estructuras de datos. Por otro lado, los usuarios normales con acciones limitadas.

Para dar persistencia a los roles de los usuarios de cada proyecto/lista, se ha utilizado el campo users. Este campo se encuentra en los documentos de la base de datos que representan estas estructuras. Se representa como un array de estructuras donde cada una de estas estructuras, tiene dos valores. El nombre del usuario asignado y el rol de dicho usuario en el proyecto/lista correspondiente.

Tanto en los proyectos como en las listas, el usuario que añade el nuevo proyecto/lista es el que recibe el rol de administrador directamente.

También hay que destacar que un mismo proyecto/lista puede contar con varios administradores simultáneamente. Desde la pantalla de configuración del proyecto/lista, un administrador puede editar el rol de todos los usuarios asignados. Esta funcionalidad puede observarse en la figura 5.7.

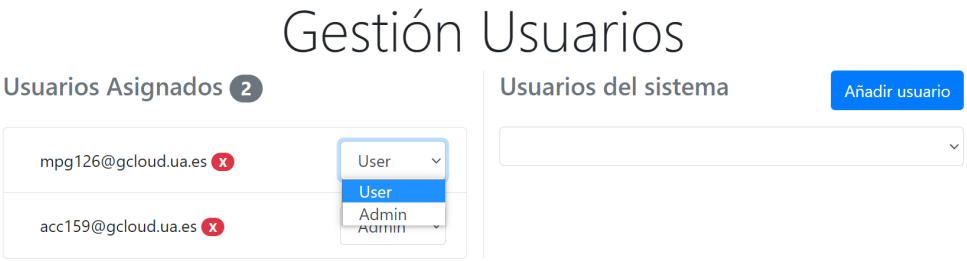

Figura 5.7: Edición del rol de un usuario en una lista o proyecto

Cualquier proyecto o lista, requiere en todo momento de un usuario con el rol de administrador. Este factor ha conllevado que se haya optado por la decisión de no cifrar la información respecto a los roles de los usuarios para los proyectos/listas.

Al no estar cifrados, si un administrador decide ser eliminado de un proyecto/lista, desde el servidor se puede llevar a cabo un procedimiento para establecer un nuevo administrador.

Cuando el servidor recibe solicitud de eliminación de un usuario administrador, se revisa si para el proyecto/lista correspondiente quedan o no más usuarios. En caso de no quedar más usuarios, se procede a eliminar el proyecto/lista entero, junto a todas las estructuras inferiores asociadas al elemento a eliminar. En caso contrario se revisa si ya existe algún usuario con rol de administrador, dándose dos situaciones:

- En caso de que lo anterior sea afirmativo, no se realiza ninguna acción más.
- De no ser así, se escoge al primer usuario en el array de usuarios, para que sea el nuevo administrador.

El proceso está representado en un diagrama de flujo en la figura 5.8.

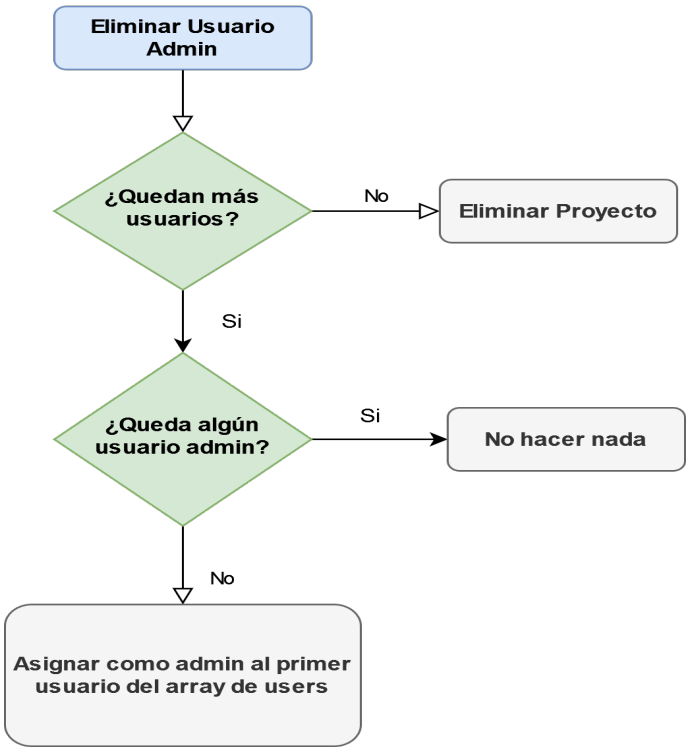

Figura 5.8: Diagrama de flujo del borrado de un usuario administrador de un proyecto o lista

En los siguientes subapartados, se detallarán las diferentes funcionalidades de los dos tipos de roles en cada una de las estructuras:

5.6.6.1 Roles en los proyectos

En lo referente a los proyectos, los usuarios administradores disponen de una serie de funcionalidades particulares:

- Capacidad de añadir listas al proyecto.
- Poder editar la configuración del proyecto, es decir, poder editar los campos (nombre, descripción) de un proyecto.
- Posibilidad de eliminar todo el proyecto, junto a las listas y las tareas correspondientes.
- Visualizar las listas del proyecto en las que este asignado.

El resto de los usuarios tan solo podrán limitarse a la última funcionalidad, es decir, poder visualizar las listas en las que estén asignados.

Es importante destacar que un administrador de un proyecto no podrá visualizar listas en las que no este asignado. Dado que rompería el esquema de buscar garantizar la máxima privacidad de los datos del sistema.

En la figura 5.9 se puede observar un caso de uso para ver, de manera gráfica, las diferentes funcionalidades que puede efectuar cada tipo de usuario sobre un proyecto.

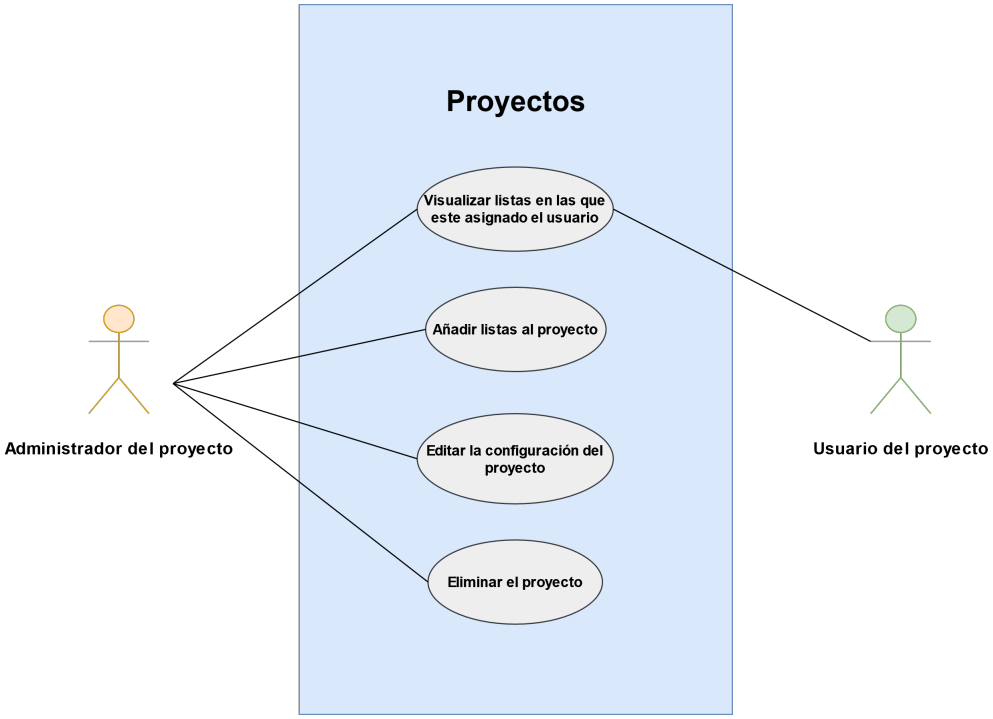

Figura 5.9: Caso de uso de las funcionalidades de los roles en un proyecto

5.6.6.2 Roles en las listas

En lo referente a las listas, aquí hay una mayor variedad de funcionalidades para los usuarios normales. Estos podrán realizar lo siguiente:

- Visualizar las tareas de una lista.
- Añadir tareas a la lista.
- Editar las tareas de la lista.
- Eliminar las tareas de la lista.

Como se puede ver, cualquier usuario normal dentro de una lista podrá hacer cualquier tipo de operación a nivel de tarea, pero no de lista. Por otro lado, los usuarios administradores de la lista dispondrán de todas estas funcionalidades mencionadas anteriormente, junto a las siguientes:

- Editar la configuración de la lista.
- Eliminar la lista.

Todas estas funcionalidades se pueden ver en la figura 5.10 que representa al caso de uso de los distintos roles sobre las listas.

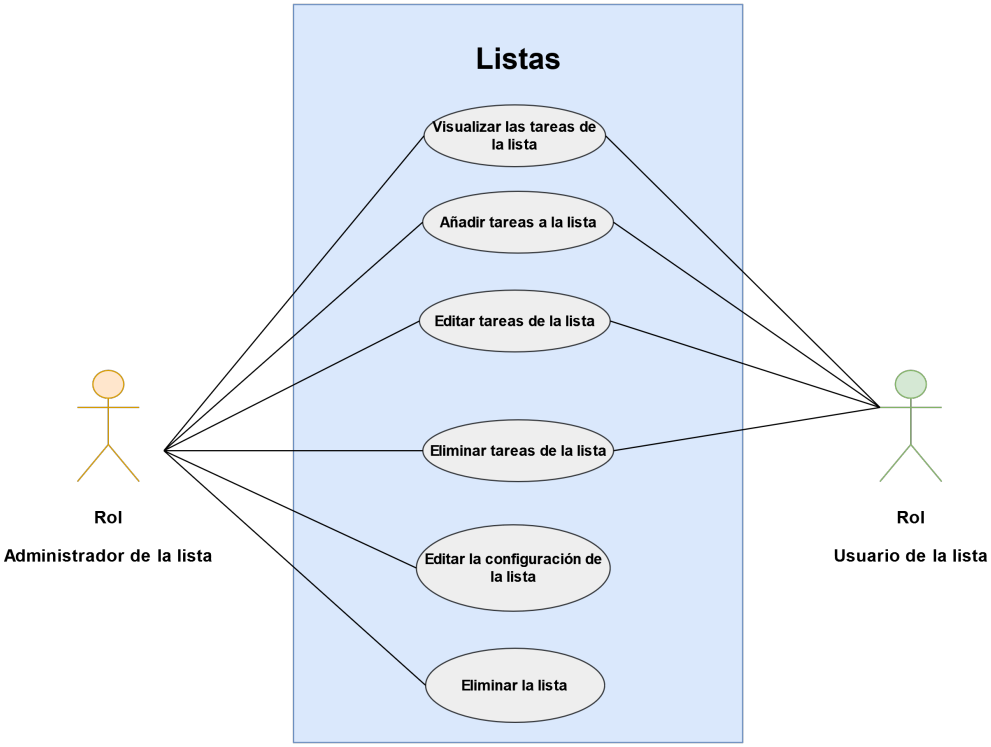

Figura 5.10: Caso de uso de las funcionalidades de los roles en una lista

5.6.7 Seguridad en las comunicaciones

En las dos subsecciones siguientes, se detallan las medidas desarrolladas para dotar de mayor seguridad a las comunicaciones.

5.6.7.1 HTTPS

Los datos enviados desde el cliente ya se encuentran cifrados, mediante el procedimiento comentado anteriormente. A pesar de esto, se ha optado por utilizar TLS sobre el protocolo HTTP, para poder así realizar todas las comunicaciones con HTTPS.

De esta manera se logra dotar de una doble capa de seguridad a la información transmitida. Garantizando una total confidencialidad en caso de que una tercera parte intercepte la información.

Para lograr establecer un servidor HTTPS, se hace uso del enrutador *mux*. Este facilita la tarea, ya que simplemente se le debe indicar que se quiere establecer un servidor TLS, junto a la dirección y el puerto de escucha. Junto a estos datos también es necesario indicarle un certificado digital y una clave privada.

Tanto el certificado como la clave privada han sido generados mediante la herramienta OpenSSL. Esta herramienta permite llevar a cabo la generación de certificados autofirmados,

con la correspondiente clave privada asociada a la clave pública del mismo.

Se ha optado por el uso de un certificado autofirmado, dado que el proyecto se considera que su desarrollo ha terminado en un entorno de preproducción. En caso de trasladarlo a producción en un entorno real, lo ideal sería un certificado firmado por una Autoridad Certificadora de confianza.

Tras generar tanto el certificado *server.crt* como la clave privada *server.key* mediante OpenSSL. Estos se almacenan en distintos ficheros en una carpeta *certs* en el directorio del servidor.

Por último, mencionar que no se utiliza el certificado ni la clave privada de la AC implementada, por motivo de mantener separados, a nivel conceptual, el servidor de la autoridad certificadora.

5.6.7.2 JWT

Además del empleo de TLS que dota de confidencialidad a la información transmitida en las comunicaciones. Se ha implementado un mecanismo que proporcione una capa de autorización de los clientes sobre la API del servidor. El mecanismo escogido ha sido JWT (*JSON Web Token*).

El funcionamiento de los tokens es muy sencillo. Una vez un usuario se ha autenticado en el sistema de manera exitosa mediante el login, se le proporciona un token. Dicho token tiene una serie de campos, entre los que se encuentra el tiempo de expiración de este.

Por motivos de garantizar la seguridad se ha optado por limitar este tiempo a 10 minutos. De esta manera si una tercera entidad consigue interceptar un token, dispondrá de un periodo de tiempo más reducido para suponer una amenaza. El tiempo de expiración de los tokens es configurable para ajustarlo en función del nivel de seguridad que se quiera alcanzar.

En cada petición que se realice a la API del servidor, exceptuando las peticiones a la ruta de registro e inicio de sesión, deberá adjuntarse un token válido (en la cabecera *Authorization* de la petición HTTP). En el lado del servidor, existe un middleware encargado de validar tanto que el token es válido, como la vigencia de este.

En caso de que una de estas dos condiciones no se den, se notificaría al cliente. Por otro lado, en caso afirmativo se procesará la petición, devolviendo, además de la respuesta pertinente, un nuevo token de refresco en esta.

En cada una de las peticiones que efectúa un cliente al servidor se produce un refresco del token, de esta manera disponer de tokens con un tiempo tan reducido no supone un problema.

También hay que mencionar que en el cliente antes de enviar el token se produce un chequeo para verificar que el mismo aún tiene vigencia.

Además de esto, en el cliente, tanto en la pantalla principal de la interfaz (*Home*), como en la interfaz que muestra las tareas de una lista. Cuando un usuario no realice ninguna acción en un plazo de 9 minutos, es decir, a falta de tan solo un minuto para la expiración del token. Se le indicará al usuario que el token del que dispone está a punto de expirar.

Al indicárselo, se le ofrecerá o bien cerrar la sesión directamente o solicitar un token de refresco, como puede visualizarse en la figura 5.11. En este último caso se efectuará una petición al servidor. Devolviendo este un token de refresco para el usuario dado, otorgándole así un nuevo plazo de 10 minutos con el que trabajar.

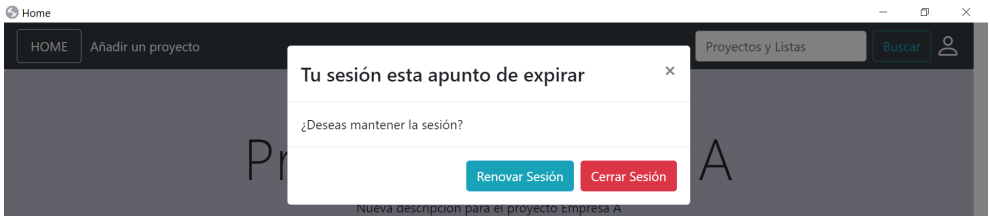

Figura 5.11: Modal indicando al usuario si desea renovar su token

5.6.8 Traza de logs

La última medida de seguridad tomada es la incorporación de un sencillo sistema de logs. Gracias a este se puede llevar a cabo una traza de todos los eventos registrados por la API del servidor.

El sistema de logs registra todas las peticiones que se realizan a la API, dejando constancia de diversos datos. Se registra la dirección IP del cliente que efectúa la petición junto al puerto de origen. También se registra el método empleado en la petición (GET, POST, PUT o DELETE) y la ruta de la petición.

Para dar persistencia a estos logs, se guardan como documentos de la colección (*logs_server*) de la base de datos como se comentó en el apartado 5.3.1. De esta manera se puede consultar la traza de los logs en cualquier momento.

La información que se refleja en los documentos de la colección, es la que se ha comentado anteriormente. Un ejemplo de un documento de la colección puede visualizarse en la figura 5.12

Figura 5.12: Documento de la colección de logs de la base de datos

5.7 Dificultades/Problemas encontrados

De cara al desarrollo del sistema propuesto, la gran mayoría de complicaciones encontradas han sido referentes al aprendizaje inicial del lenguaje de programación GO, dado que ha sido la primera ocasión en la que se empleaba este.

Otro aspecto costoso al principio del desarrollo, fue comprender como utilizar la librería *lorca*. También el añadir el empleo de la librería *html/template* al proyecto, conllevo algunas complicaciones de cara a poder terminar mostrando a los usuarios los datos de la mejor manera posible. Una vez ya se conoce la manera de poder realizar bucles, condicionales y demás, resulta sencilla su utilización.

Con respecto a la parte criptográfica, dada la facilidad y buena documentación que existe sobre las librerías empleadas para lograr aplicar los algoritmos y técnicas criptográficos deseados, no ha sido costoso hacer las implementaciones de estos. Lo más complicado fue poder implementar la parte de la autoridad certificadora y ejecutar las validaciones de los certificados de los usuarios.

En lo referente al sistema de roles y al usuario *admin*, ambos aspectos no se terminaron de definir con exactitud en las primeras fases y esto conllevo tener que efectuar algunas modificaciones en fases posteriores. Pese a ello, debido a la versatilidad de las bases de datos *NoSQL* como MongoDB, fue finalmente sencillo incorporarlos.

Los JWT también necesitaron algunas modificaciones respecto al planteamiento inicial con el que se utilizaban. Pero finalmente se ha logrado incorporarlos de manera satisfactoria.

6 Resultados

El proyecto desarrollado se encuentra disponible en mi repositorio de GitHub (Adrián, Caparrós Carrillo, 2022-05-25). Este se ha completado cumpliendo con todos los objetivos iniciales que se propusieron. Los resultados logrados son visibles de cara a los usuarios finales, solamente desde el lado del cliente.

La primera interfaz que visualizan los usuarios, es la pantalla de inicio de sesión que se puede observar en la figura 6.1. Desde esta interfaz, los usuarios pueden acceder a la pantalla de registro.

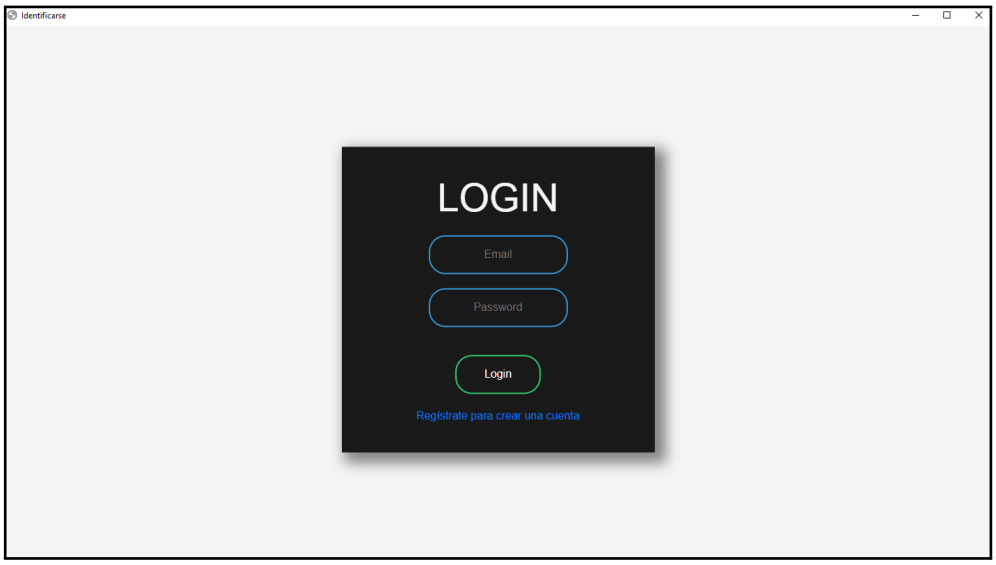

Figura 6.1: Interfaz visual de inicio de sesión

Tanto para la creación de proyectos como de listas, se han desarrollado dos interfaces muy similares. Por ejemplo, la vista de añadir un proyecto puede visualizarse en la figura 6.2. En ambas interfaces se solicitan los campos que se han definido (nombre y descripción) para estas estructuras.

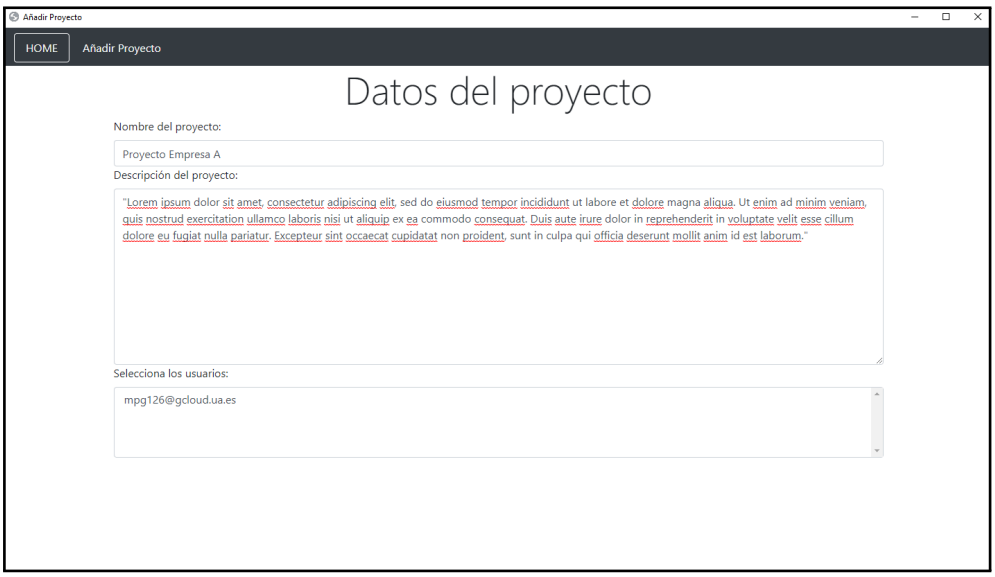

Figura 6.2: Interfaz visual de la creación de un proyecto

Aunque las pantallas para añadir proyectos y listas son similares a simple vista, ambas presentan una diferencia referente a la sección de añadir usuarios al proyecto o lista que se quiere generar.

En el caso de añadir un proyecto, el usuario visualizará todos los usuarios registrados en el sistema. Para las listas tan solo se le mostrarán, para añadir, los usuarios que se encuentren asignados en el proyecto en el que se desea generar la lista.

Tanto los proyectos como las listas disponen de dos pantallas de configuración desde las que poder realizar distintas acciones.

- Editar los campos del proyecto/lista.
- Añadir o eliminar usuarios.
- Editar los roles de los usuarios.
- Eliminar el proyecto o la lista.

Los usuarios que se permiten añadir en estas pantallas de configuración siguen las mismas normas que se han comentado anteriormente, en función de si el usuario se encuentra en la pantalla de configuración de un proyecto o de una lista.

Un ejemplo de la pantalla de configuración de un proyecto puede visualizarse en la figura 6.3.

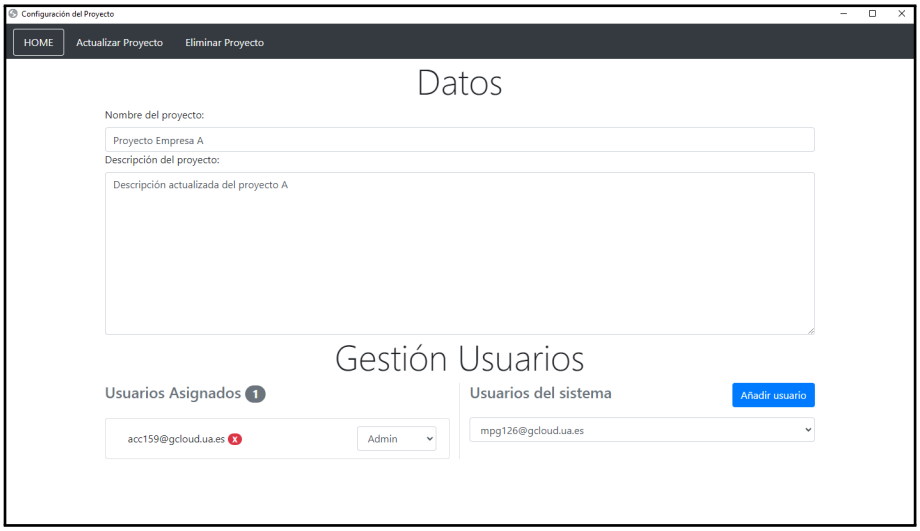

Figura 6.3: Interfaz visual de la configuración de un proyecto

En la página principal que visualizan los usuarios (ver figura 6.4), se pueden observar los distintos proyectos y listas en los que está asignado el usuario. Donde se le mostrarán, en función de si es administrador o no de esos proyectos/listas, distintos botones para las funcionalidades que ofrecen estos elementos.

Figura 6.4: Interfaz visual de la pantalla principal donde visualizar los proyectos y listas

Además, desde esta interfaz, se ofrece a los usuarios un buscador para ir rápidamente al proyecto/lista deseado. Por otra parte, en la esquina superior derecha, existe un menú de usuario que permite visualizar un resumen del total de proyectos del usuario y cerrar sesión.

De cara a las tareas, los usuarios pueden visualizar todas las tareas de una lista, como se

puede ver en la figura 6.5.

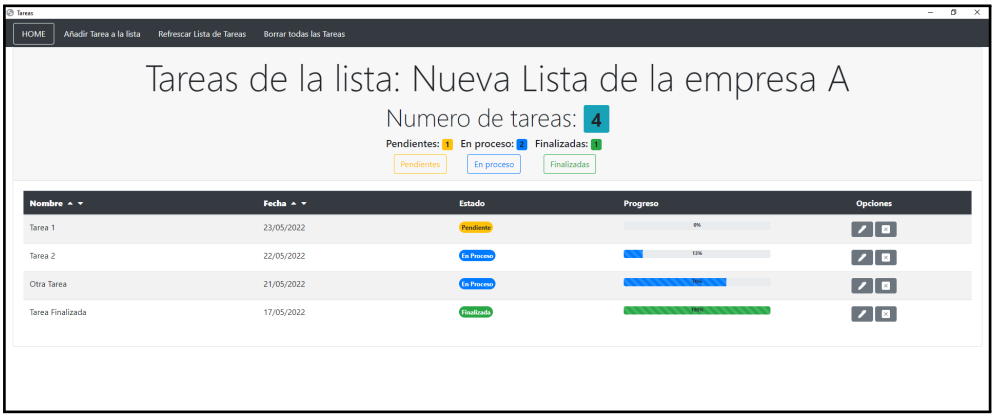

Figura 6.5: Interfaz visual de todas las tareas de una lista

En esta interfaz, se pueden ordenar de manera descendente y ascendente las diferentes tareas de una lista, en función de su nombre o de la fecha de creación. Asimismo, se pueden visualizar solamente las tareas según el estado en el que se encuentren. Se han definido tres estados: pendiente, en proceso y finalizada.

Desde esta interfaz, también es posible eliminar todas las tareas de una lista o añadir nuevas tareas.

A la hora de añadir una nueva tarea, se deben introducir una serie de campos, como se puede ver en la figura 6.6. En esta misma pantalla, se pueden adjuntar documentos o enlaces a la tarea que se está creando. Dichos elementos, pueden ser firmados en el momento en que son añadidos.

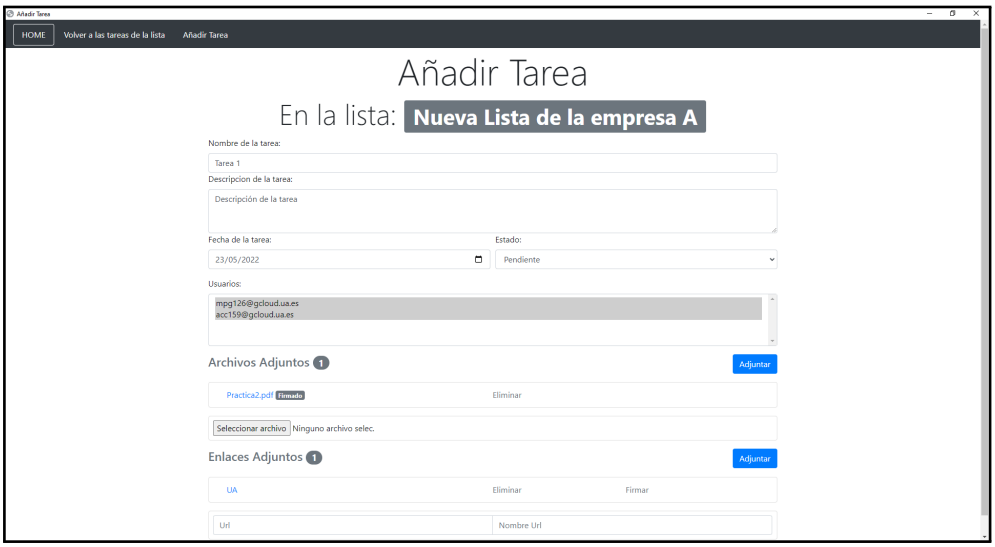

Figura 6.6: Interfaz visual de añadir una tarea

Una vez añadida una tarea, también es posible acceder a la misma, para ver los detalles de esta o incluso realizar modificaciones. Esta interfaz puede visualizarse en la figura 6.7.

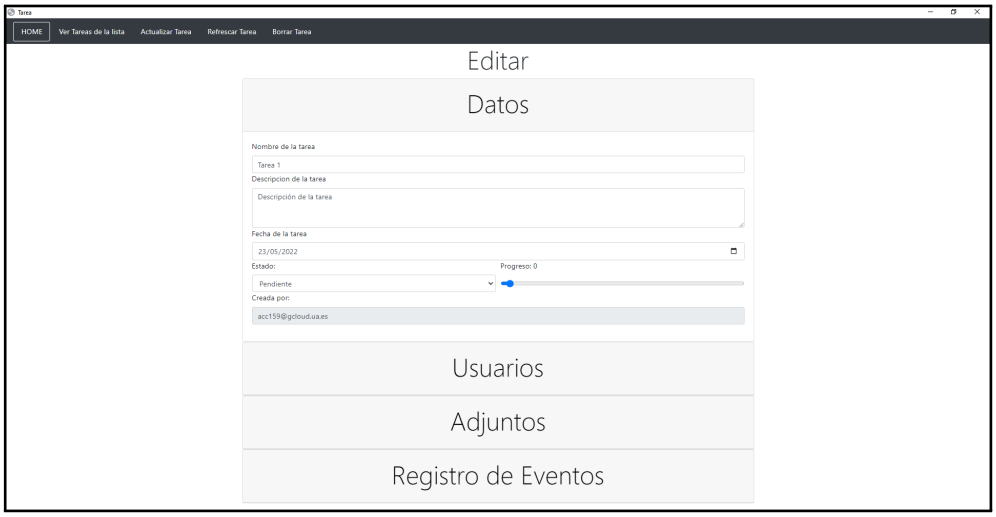

Figura 6.7: Interfaz visual de la configuración de una tarea

Como se puede observar en la figura mencionada, en esta interfaz se dan 4 secciones diferenciadas:

- Los datos de una tarea, totalmente editables.
- Los usuarios asignados a la tarea; solamente se pueden asignar usuarios miembros de la lista.
- Los elementos adjuntos de la tarea, que se pueden firmar o no.
- El registro de eventos, que puede visualizarse en la figura 6.8. En este se pueden llevar a cabo las firmas de los eventos asociados a la tarea.

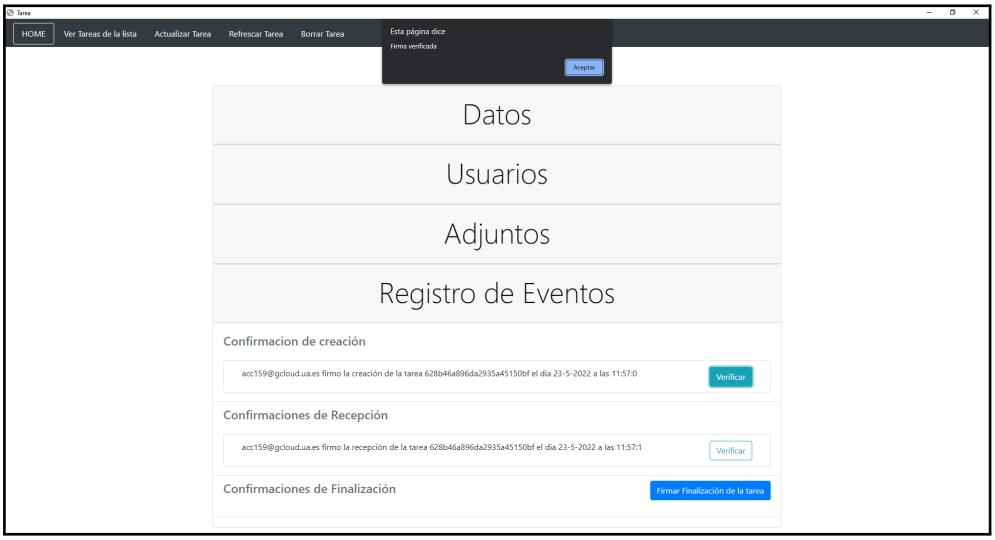

Figura 6.8: Registro de eventos de una tarea

Como se puede ver en la figura mencionada, el usuario puede firmar y validar los distintos eventos relacionados con una tarea.

Destacar que cuando un usuario efectúe cambios, en cualquiera de las tres estructuras de datos, en caso de que no actualice dichos cambios e intente cambiar de pantalla, se le mostrará un modal para indicarle que si quiere dar persistencia a los cambios debe actualizar el elemento antes de salir. Esto se puede ver en la figura 6.9.

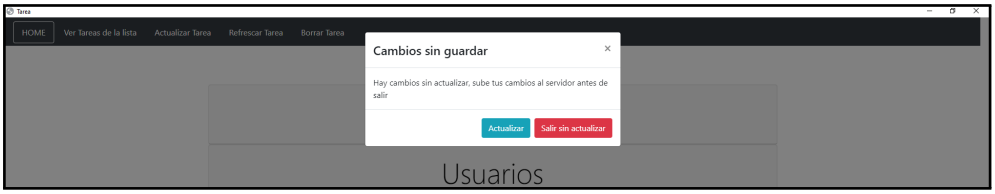

Figura 6.9: Modal indicando cambios sin guardar en el sistema

7 Conclusiones

Se ha logrado garantizar el principal enfoque del proyecto, dotar de gran fiabilidad y robustez a un sistema cliente/servidor. Teniendo en cuenta que la información en todo momento va a ser compartida por diversos usuarios.

El sistema no solo logra garantizar la confidencialidad de la información del sistema. También hace uso de la firma digital, con la cual se garantiza la autenticidad e integridad de documentos y enlaces adjuntos a las tareas, junto con los distintos eventos relacionados con estas.

Esto último, de cara al responsable de repartir las tareas en un equipo e incluso para los propios usuarios asignados a las tareas, supone una gran ventaja. Puesto que, permite conocer sin duda alguna, por ejemplo, que miembros del equipo han recibido el trabajo asignado o la finalización de una tarea.

Igualmente, en caso de darse algún problema en una tarea, el sistema es una prueba segura de los eventos que han sucedido sobre esta gracias a garantizar el no repudio.

Sin olvidar mencionar que todo el sistema de firmas, cuenta con la validez que ofrecen los certificados digitales.

7.1 Líneas de mejora

Pese al cumplimiento de los objetivos iniciales, las líneas de mejora en cuanto a funcionalidades son infinitas debido a la naturaleza del proyecto. Desde ampliar los campos de las estructuras de datos, hasta añadir diferentes y variadas formas de visualizar las tareas de una lista, entre otros.

Algunas de las líneas sobre las que se quiere hacer énfasis son las siguientes:

Emplear correos electrónicos de Google a la hora de registrar a los usuarios, para poder hacer uso de la API de Google Drive. Esto permite que en caso de que un usuario sea bloqueado, se le deshabilitaría de todos los enlaces de Google Drive que tenga compartidos en las tareas. De esta manera, dicho usuario, no podría acceder más al contenido de cualquier enlace que se haya compartido en las listas de tareas, en las que se encuentre asignado.

Otra mejora relacionada con la seguridad sería la incorporación de un sistema de doble factor de autenticación (2FA). Con el cual dotar de mayor seguridad al inicio de sesión de

los usuarios, permitiendo dar una capa de seguridad extra, a uno de los puntos claves del sistema. Esta solución se podría aplicar también, para reforzar y proteger la realización de las firmas en el sistema, solicitando un código antes de realizar cada firma. De esta manera, se lograría proteger más el uso de la clave privada del usuario con la que se firma.

En situaciones donde el sistema sea empleado por un elevado número de usuarios, se produciría una reducción del rendimiento del servidor. Con el fin de mejorarlo, se podría realizar una replicación de los servidores. Esto se podría llevar a cabo sin demasiada complicación debido a la elección de una arquitectura de tipo REST. En este tipo de arquitectura, las peticiones son sin estado (*stateless*), es decir, cada transacción es independiente. Por lo tanto, estas pueden lanzarse a distintos servidores sin ningún problema.

Esta última solución, se podría acompañar del uso de un balanceador delante de los diferentes servidores, para lograr distribuir de manera equitativa la carga entre todos los servidores.

Otra de las líneas de mejora, sería ampliar y dotar de mayor peso al sistema de logs. Se plantea incorporar los logs que se registren sobre la base de datos, o incluso desplegar un sistema de logs independiente para cada cuenta de usuario.

7.2 Seguridad frente a rendimiento

Otro punto que comentar, sería efectuar un breve análisis sobre lo que supone una aplicación en la que se hace tanto énfasis en la seguridad.

Todo tiene un coste y en el ámbito de la seguridad el mayor coste a pagar es el rendimiento del propio sistema. La seguridad y el rendimiento deben ir ligados de la mano. A pesar de esto, la tendencia años atrás solía pasar por descartar totalmente la seguridad y centrar todos los focos en el rendimiento.

Esta tendencia ha supuesto consecuencias muy graves. Los efectos de estas consecuencias, sumadas a la potencia de los ordenadores actuales, han conseguido equilibrar en cierta medida la balanza.

Las crecientes amenazas informáticas que surgen a diario, obligan a dotar a cualquier sistema de una capa de seguridad. Pese a ello, debe mantenerse en mente, en especial en sistemas que utilizaran usuarios, no descartar del todo el rendimiento del sistema.

En el caso particular de este proyecto, se ha hecho un especial énfasis en el aspecto de la seguridad. Debido a que su objetivo fundamental es el estudio e implementación de las medidas necesarias para dotar de gran seguridad a un sistema.

7.3 Reflexión final

El proyecto desarrollado se ha concebido para mostrar un enfoque acerca de como la criptografía permite dotar de mayor seguridad, en diferentes aspectos, a un sistema informático.

El software resultante del proyecto, no es un remplazo de grandes herramientas como las comentadas en este documento, es una alternativa abierta a cualquier tipo de público debido a sus sencillas interfaces y facilidad de uso. El enfoque principal de este, es proporcionar a los usuarios características relacionadas con el ámbito de la seguridad, en lugar de tratar de ofrecer un alto rendimiento.

La herramienta desarrollada es de tipo software libre. Además, proporciona a los usuarios que van a hacer uso de la misma, la seguridad de conocer con exactitud en todo momento el estado de la información, en lugar de dejarlo en manos de empresas externas. Es decir, permite que el usuario sea consciente de la seguridad y privacidad en todo momento de la información del sistema.

Las técnicas y algoritmos, que se han detallado a lo largo del documento, pueden trasladarse a distintos tipos de software. Con estas se puede proporcionar una mayor seguridad en los productos desarrollados, sin ser necesaria la aplicación de todas en conjunto, simplemente aquellas centradas en la parte que se busque ofrecer mayor seguridad.

Por último, mencionar que dentro de todo el ámbito de la ciberseguridad y las diferentes ramas que existen en esta, la criptografía supone una pieza crucial para la seguridad informática. Tanto para el presente como para el futuro que nos pueden deparar la criptografía cuántica y post-cuántica (para más información, véase García y cols., 2014-05-19; Wang y cols., 2021-10-27). Siendo vital su conocimiento teórico y en especial, disponer del conocimiento para aplicarla en diferentes tipos de sistemas, garantizando así a los usuarios la mejor seguridad.

Bibliografía

Adrián, Caparrós Carrillo. (2022-05-25). *Software desarrollado.* Descargado de https://github.com/acc159/TFG

Ataque de diccionario. (s.f.). Descargado 2022-05-26, de https://es.wikipedia.org/wiki/Ataque_de_diccionario

Auguste, K. (1883, February). La cryptographie militaire. *Journal des sciences militaires*, *IX*, 161–191. Descargado de https://www.petitcolas.net/kerckhoffs/crypto_militaire_1_b.pdf

Avast. (2022-01-13). *¿Qué es un sniffer y cómo puede protegerse?* Descargado de https://www.avast.com/es-es/c-sniffer

Bahit, E. (2016, Feb). *Scrum y eXtreme Programming para Programadores.* Descargado de http://umh2818.edu.umh.es/wp-content/uploads/sites/884/2016/02/Scrum-y-eXtrem-Programming-para-programadores.pdf

Barker, E., Chen, L., Roginsky, A., Davis, R., y Simon, S. (2019-03-21). *Recommendation for Pair-wise Key Establishment Using Integer Factorization Cryptography.* Special Publication (NIST SP), National Institute of Standards and Technology, Gaithersburg, MD. doi: https://doi.org/10.6028/NIST.SP.800-56Br2

Barker, E., y Roginsky, A. (2019-03-21). *Transitioning the Use of Cryptographic Algorithms and Key Lengths.* Special Publication (NIST SP), National Institute of Standards and Technology, Gaithersburg, MD. doi: https://doi.org/10.6028/NIST.SP.800-131Ar2

Block cipher mode of operation. (s.f.). Descargado 2022-05-26, de https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation#Counter_(CTR)

Boeyen, S., Santesson, S., Polk, T., Housley, R., Farrell, S., y Cooper, D. (2008, mayo). *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile.* RFC 5280. RFC Editor. Descargado de https://www.rfc-editor.org/info/rfc5280 doi: 10.17487/RFC5280

Breyha, W., Durvaux, D., Dussa, T., Kaplan, L. A., Mendel, F., Mock, C., … Zawodsky, P. (2018-12-21). *Applied Crypto Hardening, 24. Public Key Infrastructures.* Descargado de https://bettercrypto.org/#pkis

CCN. (2020, November). *Requisitos de Seguridad para TLS.* Descargado de https://www.ccn.cni.es/index.php/es/docman/documentos-publicos/boletines-pytec/378-pildorapytec-nov2020-seguridad-tls/file

CCN-CERT. (2021, December). *Ciberamenazas y Tendencias. Edición 2021* (Inf. Téc.). Descargado 2021-12-14, de https://www.ccn-cert.cni.es/informes/informes-ccn-cert-publicos/6338-ccn-cert-ia-13-21-ciberamenazas-y-tendencias-edicion-2021-1/file.html

Cifrado: ¿simétrico o asimétrico? (s.f.). Descargado 2019-07-18, de https://iordic.github.io/criptograf%C3%ADa/cifrado/sim%C3%A9trico/asim%C3%A9trico/seguridad/2019/07/18/cifrado-simetrico-vs-cifrado-asimetrico.html

ClickUp. (s.f.-a). *Clickup información seguridad.* Descargado 2022-05-26, de https://clickup.com/security

ClickUp. (s.f.-b). *Software clickup.* Descargado 2022-05-26, de https://clickup.com

Córdoba, D. (2018-02-02). *x509: Certificados digitales y codificaciones DER, CRT y CER.* Descargado de https://juncotic.com/x509-certificados-digitales-der-crt-cer/

Delgado, V., y Palacios, R. (2006, March-April). *Aplicaciones prácticas de la criptografía.* Descargado de https://www.iit.comillas.edu/documentacion/IIT-06-105R/Aplicaciones_pr%C3%A1cticas_de_la_criptograf%C3%ADa.pdf

Documentación driver de GO para MongoDB. (s.f.). Descargado 2022-05-26, de https://www.mongodb.com/docs/drivers/go/current/

Driver de MongoDB para GO. (s.f.). Descargado 2022-05-26, de https://github.com/mongodb/mongo-go-driver

Dworkin, M., Barker, E., Nechvatal, J., Foti, J., Bassham, L., Roback, E., y Dray, J. (2001-11-26). *Advanced Encryption Standard (aes).* Federal Inf. Process. Stds. (NIST FIPS), National Institute of Standards and Technology, Gaithersburg, MD. Descargado de https://doi.org/10.6028/NIST.FIPS.197

Dworkin, M. J. (2001). *SP 800-38A 2001 Edition. Recommendation for Block Cipher Modes of Operation: Methods and Techniques* (Inf. Téc.). Gaithersburg, MD, USA.

Framework Bootstrap. (s.f.). Descargado 2022-05-26, de https://getbootstrap.com/

Función de derivación clave. (s.f.). Descargado 2022-05-26, de https://hmong.es/wiki/Password_hash

García, A., García, F., y García, J. (2014-05-19). *Computación y Criptografía Cuántica.* Descargado de http://www.criptored.upm.es/crypt4you/temas/cuantica/leccion1/leccion01.html

García Belmont, R., Lotzin Rendón, G., Cabrera Hernández, L., Puente Pérez, M., y Verónica Méndez, O. (2018-02-02). *AES como Estándar Internacional de Cifrado.* Descargado de https://conaic.net/revista/publicaciones/Vol_V_Num1_Ene_Abr_2018/Articulo10.pdf

González Salomone, M. (2018-06-13). *Premios Fundación BBVA Fronteras del Conocimiento, "Una de las ramas más pesimistas de la ciencia".* Descargado de https://www.fbbva.es/wp-content/uploads/2018/10/NdP-Ceremonia-Fronteras-X.pdf

Grupo de Galois. (s.f.). Descargado 2022-05-26, de https://es.wikipedia.org/wiki/Grupo_de_Galois

Huguet Rotger, L., Rifà Coma, J., y Tena Ayuso, J. G. (s.f.). *Elementos de criptografía.* Descargado 2022-05-26, de https://www.exabyteinformatica.com/uoc/Informatica/Criptografia_avanzada/Criptografia_avanzada_(Modulo_2).pdf

INCIBE. (2020-07-16). *El ataque del "Man in the middle" en la empresa, riesgos y formas de evitarlo.* Descargado 2022-05-26, de https://www.incibe.es/protege-tu-empresa/blog/el-ataque-del-man-middle-empresa-riesgos-y-formas-evitarlo

INCIBE. (2022, May). *INCIBE gestiona más de 100.000 incidentes de ciberseguridad durante 2021* (Inf. Téc.). Descargado 2022-05-12, de https://www.incibe.es/sala-prensa/notas-prensa/incibe-gestiona-mas-100000-incidentes-ciberseguridad-durante-2021

Jeremiah, G., Robert, H., Petko, D. P., y Anton, R. (2007). *XSS attacks.* Descargado de https://doc.lagout.org/security/Cross%20Site%20Scripting%20Attacks%20Xss%20Exploits%20and%20Defense.pdf

Jira. (s.f.-a). *Jira información seguridad.* Descargado 2022-05-26, de https://www.atlassian.com/es/software/jira/security

Jira. (s.f.-b). *Software jira.* Descargado 2022-05-26, de https://www.atlassian.com/es/software/jira

Jones, M., Bradley, J., y Sakimura, N. (2015, May). *JSON Web Token (JWT)* (RFC). RFC Editor. Internet Requests for Comments. Descargado de http://www.rfc-editor.org/rfc/rfc7519.txt (http://www.rfc-editor.org/rfc/rfc7519.txt)

Josefsson, S., y Leonard, S. (2015, April). *Textual Encodings of PKIX, PKCS, and CMS Structures* (RFC). RFC Editor. Internet Requests for Comments.

Lenguaje de programación GO. (s.f.). Descargado 2022-05-26, de https://go.dev/

Librería Crypto de GO. (s.f.). Descargado 2022-05-26, de https://pkg.go.dev/golang.org/x/crypto

Librería jQuery. (s.f.). Descargado 2022-05-26, de https://jquery.com/

Librería JWT. (s.f.). Descargado 2022-05-26, de https://github.com/golang-jwt/jwt

Librería Lorca. (s.f.). Descargado 2022-05-26, de https://github.com/zserge/lorca

Lucena Ĺopez, M. J. (2015). *Criptografía y Seguridad en Computadores.*

Luján Mora, S. (2001). *Programación en Internet: clientes web.* CRC press.

McGranaghan, M., y Bendersky, E. (s.f.). *Go by Example: Slices.* Descargado 2022-05-26, de https://gobyexample.com/slices

Microsoft. (s.f.). *Software Visual Studio Code.* Descargado 2022-05-26, de https://code.visualstudio.com

MongoDB. (s.f.). Descargado 2022-05-26, de https://www.mongodb.com/es

MongoDB Atlas. (s.f.). Descargado 2022-05-26, de https://www.mongodb.com/es/atlas/database

Muñoz, A. (2004, September). *Criptosistema Rijndael. A Fondo.* Descargado de http://www.criptored.upm.es/guiateoria/gt_m480a.htm

Muñoz, A. (2020). *Criptografía ofensiva. Atacando y defendiendo organizaciones.*

Operador a nivel de bits. (s.f.). Descargado 2022-05-26, de https://es.wikipedia.org/wiki/Operador_a_nivel_de_bits#XOR

Paquete Context de GO. (s.f.). Descargado 2022-05-26, de https://pkg.go.dev/context

Paquete gorilla/mux. (s.f.). Descargado 2022-05-26, de https://pkg.go.dev/github.com/gorilla/mux

Paquete html/template de GO. (s.f.). Descargado 2022-05-26, de https://pkg.go.dev/html/template

Priego García, L. (2018-06-03). *Estudio del protocolo TLS (Transport Layer Security).* Descargado de http://openaccess.uoc.edu/webapps/o2/bitstream/10609/81045/6/lprigarTFM0618memoria.pdf

RAE. (s.f.-a). *Definición de Criptografía.* Descargado 2022-05-26, de https://dle.rae.es/criptograf%C3%ADa

RAE. (s.f.-b). *Definición de Criptología.* Descargado 2022-05-26, de https://dle.rae.es/criptolog%C3%ADa

Ramió Aguirre, J. (2016-02-15). *Introducción a la seguridad informática y criptografía clásica.* Descargado de http://www.criptored.upm.es/crypt4you/temas/criptografiaclasica/leccion1.html

Ramió Aguirre, J. (2020). *CFB, OFB y CTR, modos de cifra con confidencialidad.* Descargado de https://www.criptored.es/descarga/Class4cryptc4c8.4b_CFB_OFB_CTR_modos_cifra_con_confidencialidad.pdf

Ramió Aguirre, J. (2020-10-06). *Tríada confidencialidad, integridad y disponibilidad.* Descargado de https://www.criptored.es/descarga/Class4cryptc4c1.5_Triada_confidencialidad_integridad_disponibilidad.pdf

Rizzo, J., y Duong, T. (2010). *Practical Padding Oracle Attacks.* Descargado de https://dl.packetstormsecurity.net/papers/database/padding-oracle.pdf

Shannon, C. E. (1949). Communication theory of secrecy systems. *The Bell System Technical Journal*, *28*(4), 656-715.

Software Git. (s.f.). Descargado 2022-05-26, de `https://git-scm.com/`

Software Github. (s.f.). Descargado 2022-05-26, de `https://github.com`

Software Google Meet. (s.f.). Descargado 25 Mayo 2022, de `https://meet.google.com/`

Software MongoDB Compass. (s.f.). Descargado 2022-05-26, de `https://www.mongodb.com/es/products/compass`

Software Postman. (s.f.). Descargado 2022-05-26, de `https://www.postman.com/`

Trello. (s.f.-a). *Software trello.* Descargado 2022-05-26, de `https://trello.com`

Trello. (s.f.-b). *Trello información seguridad.* Descargado 2022-05-26, de `https://trello.com/es/legal/security`

Wang, J., Hayes, B., y NIST. (2021-10-27). *Post-Quantum Cryptography: A Q&A With NIST's Matt Scholl.* Descargado de `https://csrc.nist.gov/Projects/post-quantum-cryptography`