Escuela
Politécnica
Superior

Contabilidad Segura

Grado en Ingeniería Informática

Trabajo Fin de Grado

Autor:
Carlos Ascó Rico
Tutor/es:
Jose Vicente Aguirre Pastor
Antonio Zamora Gomez

Junio 2021

Universitat d'Alacant
Universidad de Alicante

Contabilidad Segura

Programa cliente/servidor usando técnicas criptográficas.

Autor
Carlos Ascó Rico

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

Grado en Ingeniería Informática

Escuela
Politécnica
Superior

Universitat d'Alacant
Universidad de Alicante

ALICANTE, Junio 2021

Preámbulo

Las razones principales del desarrollo de este proyecto han sido entre otras:

- El estudio de técnicas criptográficas para la protección de información.
- Desarrollo de una aplicación desde 0.

La elección de realizar una aplicación de contabilidad viene dada por el querer adquirir conocimientos más allá de lo visto en el grado. Aprovechando que mi padre trabajó de contable, quise realizar una investigación de los conceptos más básicos de una contabilidad y aplicarlos en el proyecto. Bien es sabido que el tema económico puede ser objeto de robo lo que me llevó a pensar que usar técnicas de cifrado para ocultar la información sensible puede ser una idea interesante.

Gracias a la realización de este trabajo he podido aprender más acerca de lo que sería crear un proyecto desde 0, con las fases de desarrollo que conlleva, aunque sea a corta escala.

Agradecimientos

En primer lugar querría agradecer a mis tutores Antonio Zamora y en especial a Jose Vicente Aguirre su trabajo y dedicación para ayudarme a conseguir mis objetivos. Gracias a sus lecciones tanto en el grado como en este trabajo he crecido como ingeniero y también como futuro profesional. Cada consejo me ha ayudado a aprender y madurar en mis análisis. Gracias por acompañarme en esta última etapa.

Un lugar muy especial tienen mis amigos de carrera. No me quiero dejar a ninguno de ellos. Tonet, Cayetano, Mateo, Arturo y Jordi. Estos años han sido de los mejores que he tenido. Con ellos he aprendido lo que cuesta conseguir lo que deseas.

También agradecer a mis padres que siempre me han apoyado en todo momento. Me habría sido imposible hacer tanto el grado como este proyecto sin su ayuda. Sois mi motivo de orgullo.

Para concluir, mencionar a quien ha sido mi motor todos estos años. Ainara, si estoy aquí es gracias a ti. Siempre has sido mi ejemplo y más si hablamos de dedicar esfuerzo a lo importante. El haber conseguido mis metas solo ha sido por seguir tus pasos.

A todos ellos dedico mi trabajo.

La ciencia de crear códigos secretos se llama criptografía y la ciencia de romperlos criptoanálisis.
"Criptonomicón" (1999), Neal Stephenson

Índice general

Índice de figuras

Índice de tablas

1 Introducción

Contabilidad Segura es el proyecto de software que se desarrolla en este trabajo. Para ello se toma como base la arquitectura cliente/servidor. Además se utilizarán técnicas criptográficas para la de ocultación de información sensible. Estas técnicas criptográficas serán las verdaderas protagonistas del proyecto. Estas serán aplicadas también para establecer una comunicación segura, tanto en el canal como a la hora de otorgar la información al usuario que disponga de los permisos pertinentes.

El escenario para implementar todas estas técnicas será el de poder gestionar la contabilidad de una empresa pequeña, añadiendo la estructura de datos básica para el almacenamiento de los diferentes apuntes y asientos contables.

Para el desarrollo del proyecto se ha realizado un análisis de los requerimientos, tanto de las estructuras de datos como del diseño de software, estudio y elección de los sistemas criptográficos más adecuados para segurizar la información y por último la implementación de los mismos.

1.1 Porqué Contabilidad

Es sabido que la información es poder, por ello desde la antigüedad siempre se ha intentado ocultar la información. La información siempre puede ser sensible según el contexto. Centrándonos en la pequeña/mediana empresa podríamos decir que el capital podría ser información con dicha sensibilidad.

A día de hoy las pequeñas empresas son un objetivo de los ciberdelincuentes. Uno de los ataques que pueden llegar a realizarse a las pequeñas empresas es, mediante *phising* [1], cifrar la información contable y otorgar la clave de descifrado a cambio de una recompensa económica. El chantaje puede ser la venta de esta información a otras empresas competidoras o simplemente exponer sus cuentas al público.

La ocultación de información en un contexto real como en el de la contabilidad de una empresa puede ser la clave de la seguridad para una empresa. Es por ello que la investigación de este proyecto se basa en este tema concreto, además de querer investigar sobre el campo de la criptografía y como aplicar esta para la seguridad de aplicaciones informáticas en internet.

[1] Estafa cuya meta es obtener información sensible de la víctima que pueden ser desde cuentas personales/o coorporativas de redes sociales hasta cuantas bancarias.

Otro motivo por el cual se ha elegido la contabilidad como temática para este proyecto es la falta de aplicaciones de software libre que exploren las opciones de criptografía más allá de cifrar el canal.

Es por ello que se realizara un estudio de que técnicas criptográficas podremos usar para asegurar en la medida de lo posible la segurización de la información.

1.2 Porqué Cliente/Servidor

El escenario del cual partimos sería el de una aplicación en el que tendríamos por una parte el administrador del sistema y por otra los encargados contables como usuarios.

Hoy en día la deslocalización de las empresas es una realidad y aunque puede ser un problema, si nos centramos en temas de seguridad, hubiéramos optado por una aplicación de escritorio (local), ya que es evidente que otorgaría la seguridad añadida. Los datos no viajarían a través de internet y gracias a esto no pueden ser robados por un tercero.

Sin embargo si nos ceñimos a la realidad de muchas empresas es algo que actualmente sería un paso atrás, ya que implicaría un retraso, y por ello el coste adicional, a la hora de usar la aplicación por un experto.

Gracias a las técnicas de ocultación de información podremos crear un canal seguro que impida en la mayor parte de los casos obtener esa información de forma fraudulenta. De esta manera tendríamos a los usuarios trabajando desde las diferentes sedes y tratando directamente con el servidor para la gestión de sus apuntes contables.

1.3 La información a proteger

Como bien se ha descrito en el primer apartado, toda la información puede ser sensible. En este proyecto se ha propuesto el ocultamiento de todos los ejercicios contables como información sensible, por ello se usarán técnicas de cifrado sobre esta parte de la aplicación, englobando toda la contabilidad de una empresa.

Además de esto segurizaremos el canal de comunicación usando técnicas como HTTPS y tokenización que nos darán una mayor seguridad ante ataques que podrían darse como la modificación de dicho canal con fines delictivos.

Por último daremos la posibilidad de firmar los listados contables con nuestro certificado digital desde el cliente para corroborar la autenticidad de los documentos creados.

2 Marco Teórico

Antes de profundizar en la realización de trabajo veremos los diferentes puntos clave de este para poder entender con mayor claridad las técnicas criptográficas así como los aspectos más relevantes de este.

2.1 Cifrado de información

2.1.1 Qué es cifrar

Entendemos por cifrar como la ocultación de información mediante un código que pueden ser guarismos, letras o símbolos, ligados a una clave.

Desde la antigüedad se ha estudiado la manera de realizar de manera correcta la ocultación de información, el nombre de esta ciencia es criptología.

Podríamos decir que la criptología es el estudio de la criptografía y el criptoanálisis.

- Criptografía: Ciencia en la que la principal motivación es crear métodos de cifrado y descifrado, los cuales vean aumentado el tiempo necesario de su uso cuando este lo es ilegítimo, ya sea para obtener o modificar la información que se intenta ocultar. Esto dicho de forma más coloquial seria que el tiempo invertido en romper dichos sistemas criptográficos siempre sea mayor a uso correcto de estos.
- Criptoanálisis: Ciencia que estudia la fiabilidad de los sistemas criptográficos. Para ello se usará el tiempo como medida a la hora de valorar cuan fiable es un criptosistema intentando obtener información cifrada sin conocer la clave.

En la historia antigua se crearon muchos criptosistemas para que la información que se quería transmitir no fuera comprometida. Las 2 técnicas que se usaron fueron la difusión y la confusión.

- Difusión: El mensaje será redistribuido con transposiciones para que de esta manera el mensaje se disperse y gracias a las propiedades estadísticas del lenguaje sea difícil su descifrado sin la clave.
- Confusión: El mensaje será modificado con sustituciones de sus componentes para que de esta manera el par clave de cifrado/texto cifrado sea lo más compleja que se pueda.

Ejemplos de difusión los podemos ver en la **scitala espartana** (ver **figura 2.1**). En este criptosistema se enrollaba una cinta en la cual escribiríamos el mensaje una vara con un grosor específico. El grosor sería la clave de cifrado y descifrado. El texto debería ser escrito pasando

por las diferentes secciones de la cinta de manera horizontal de manera que al desenrollar la cinta todas las letras quedaran desordenadas. Para poder leer el mensaje necesitaríamos la vara con el grosor correcto, ya que si no era así no podríamos ordenar las letras de manera correcta.

Figura 2.1: Scitala Espartana.

Por otra parte ejemplos de confusión lo tenemos en el **cifrado de Julio Cesar**. Su funcionamiento se basaba en que todas las letras del abecedario iban a sustituirse por la tercera consecutiva.

2.2 Cifradores modernos

Podríamos atribuir su nacimiento a la necesidad de ocultar información en tiempo de guerra, más concretamente en plena Segunda Guerra Mundial, el proyecto al cual se debe es el llamado "Ultra", en el cual se encontraban varios científicos, entre otros Alan Turing. Ellos lograron descifrar la máquina Enigma gracias a Colossus (ver **figura 2.2**) el primer computador de la historia. Desde ese momento la historia de la informática ha ido muy ligada a la de la necesidad de ocultar información sensible. Muchas de estas investigaciones han ido de la mano del aparato militar como por ejemplo investigaciones de la NSA (Agencia Nacional de Seguridad en Estados Unidos)

En la actualidad, ahora con un mercado totalmente informatizado, la importancia de la ocultación de información sensible es uno de los mayores aspectos a tener en cuenta en lo que a seguridad se refiere.

En los últimos años los ciberdelincuentes han puesto su punto de mira en la pequeña y mediana empresa. Estas están menos acostumbradas a darle la importancia que merece a mantener sus datos segurizados y que sus canales de información sean seguros.

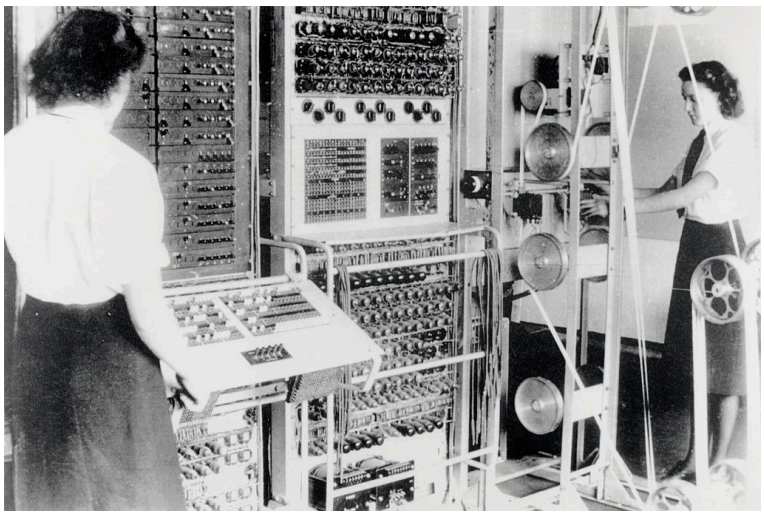

Figura 2.2: Computadora Colossus.

2.3 Tipos de cifrado

Un criptosistema siempre va a cumplir estas expresiones:

$$E_k(m) = c$$

$$D_k(c) = m$$

A continuación analizaremos cada una de las variables:

- m : Sería el texto en plano el cual va a ser objeto de cifrado o descifrado según sea el caso. Haría referencia al conjunto de mensajes en claro.
- k : Sería la clave que vamos a usar. Haría referencia al conjunto de claves.
- c : Sería el criptograma, texto cifrado. Haría referencia al conjunto de textos cifrados
- E : Sería la función de cifrado. Esta haría referencia al conjunto de funciones de cifrado.
- D : Sería la función de descifrado. Esta haría referencia al conjunto de funciones de descifrado.

Podríamos concluir las expresiones anteriores de la siguiente manera:

$$D_k(E_k(m)) = m \qquad \forall m \in M$$

Se puede diferenciar a los criptosistemas en simétricos y asimétricos.

2.3.1 Cifrado simétrico

Estos criptosistemas tienen la peculiaridad de usar la misma clave tanto en el cifrado como en el descifrado. Por ello suelen ser más fáciles de usar. La fortaleza del sistema se verá principalmente relacionado a la longitud de su clave.

Para que esta clave sea fiable se tendrían que dar los siguientes requisitos:

- Solo con el criptograma no se podrá obtener el texto en claro ni la clave
- El tiempo necesario para obtener la clave sabiendo el texto en plano y su criptograma debe ser mucho más alto que descifrar con la clave.

Como se puede deducir el mayor peligro de este tipo de criptosistemas son dos factores:

- Cantidad elevada de claves para comunicaciones seguras. Para cada par de usuarios necesitaremos de una clave diferente.
- Compartir las claves entre usuarios. Es el hecho de que para cifrar y descifrar se debería compartir esta clave y al ser una única clave cabe la posibilidad de que esta se vea vulnerada.

2.3.1.1 Tipos de cifrado simétrico.

Podríamos distinguir entre 2 tipos, cifrador en flujo y en bloque.

Los cifradores en flujo (ver **figura 2.3**) serian aquellos métodos de cifrado que usan un criptosistema en el que se combinaría el texto en plano con un flujo de cifrado pseudoaleatorio (secuencia cifrante). Será el resultado de pasar dichos datos como representación de bit por una XOR.

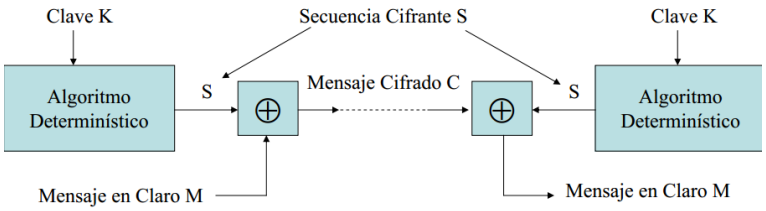

Figura 2.3: Cifrador en Flujo.

Los cifradores en bloque, los cuales usaremos en el proyecto, serian aquellos en los que el texto en claro sería cifrado agrupando los símbolos en grupos de 2 o más elementos.

Como características generales podríamos indicar las siguientes

- Cada símbolo se cifra de manera dependiente a sus adyacentes
- Cada unos de estos bloques siempre se cifrara de la misma manera
- Siempre que se use un texto en claro y clave se obtendrá el mismo criptograma.

- Podremos descifrar independientemente los bloques. No sería necesario por lo tanto descifrar el mensaje entero.

En este proyecto usaremos el algoritmo AES.

2.3.1.2 AES

En 1996, el Instituto Nacional de Estándares y Tecnología (NIST) convoco un concurso internacional para el desarrollo de un nuevo algoritmo que sea el sustituto de DES(Data Encryption Standard). El motivo fue la ya probada debilidad de esto gracias al DES Challenge[1].
El nombre de este nuevo algoritmo sería llamado AES(Advanced Encryption Standard). Este concurso lo gano el algoritmo Rijndael (acrónimo formado por los nombres de sus creadores, Vicent Rijmen y Joan Daemen).

Las características generales de AES serian las siguientes:

- Las longitudes de los bloques de texto son de 128 bits.
- La clave podría ser de 128, 192 o 256 bits.
- La matriz de estado, la cual será en la que se realizan todos los cálculos del algoritmo sería una de tamaño 4 x 4, cada una de estas celdas con un tamaño de un byte.
- Los cálculos que se usaran serán tanto técnicas de sustitución como de permutación y polinómicas.

2.3.1.3 Esquema general de AES

A diferencia de su antecesor DES, cambiaremos la red Feistel por una red de sustitución-permutación. Esto le dará una mayor velocidad y además requerirá de poca memoria para su ejecución.

Como podemos ver en la **figura 2.4** el pseudocódigo estaría divido en 3 partes: Inicio, Rondas y Final. El número de rondas que tendremos que realizar dependerá de la clave de cifrado elegida siendo 9 para una clave de 128 bits, 11 si es de 192 bits o 13 para una clave de 256 bits.
Las operaciones que realiza el algoritmo son las siguientes:

- AddRoundKey: Operación que usaremos para crear las subclaves que necesitaremos para todas las rondas del algoritmo. Esta función realizará un XOR con cada byte entre el texto en claro y la clave. Ver **figura 2.5**.
- SubBytes: Se realizará la sustitución de cada byte de la matriz de estado por el contenido de una tabla de búsqueda base. Ver **figura 2.6**.

[1]Los Desafíos DES fueron una serie de concursos de ataques de fuerza bruta creados por RSA Security para resaltar la falta de seguridad proporcionada por el Estándar de cifrado de datos.

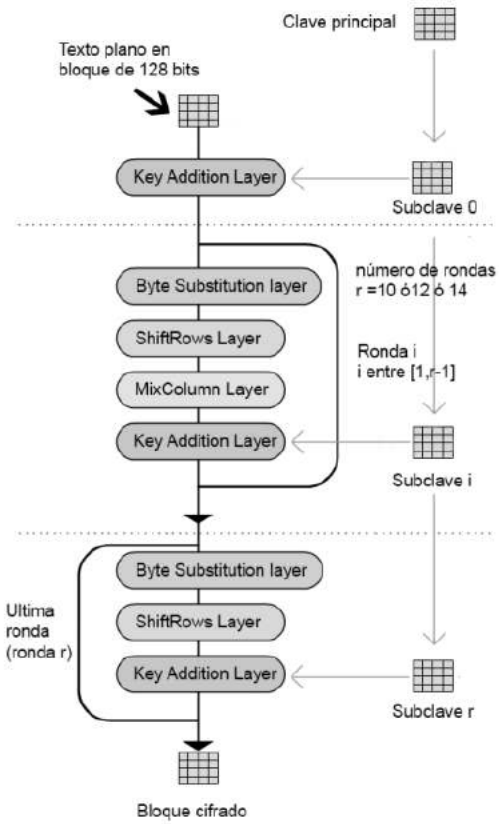

Figura 2.4: Esquema general AES.

- ShiftRows: Se realizará una permutación de la matriz de estado siendo esta que la 2 fila rotara 1 vez, la 2 fila rotara 2 veces y la última fila rotara 3 veces. Ver **figura 2.7**.
- MixColumns: Se realizará una multiplicación polinómica en el campo de Galois

$$GF(2^8)$$

de un polinomio fijo por cada columna de la matriz de estado. Ver **figura 2.8**.

2.3.2 Cifrado asimétrico

La diferencia con los sistemas simétricos es que esta vez usaremos 2 claves, una para el cifrado de datos y otra para el descifrado de lo cifrado. Este tipo de cifrados también son llamados cifrados de clave pública. El motivo es porque una será pública (generalmente para usarla en el cifrado) y la otra sería privada (para usarla en el descifrado). Ver **figura 2.9**.

En este proyecto usaremos el cifrado asimétrico tanto para cifrar información como para el uso de firmas electrónicas. Para ello usaremos el algoritmo RSA tanto en el cifrado de información como en el uso de certificados digitales.

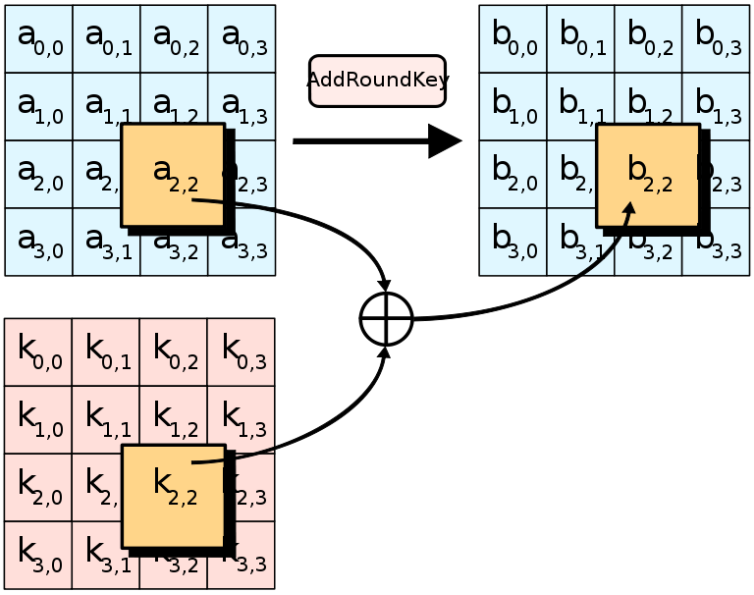

Figura 2.5: AES - AddRoundKey.

2.3.2.1 RSA

RSA fue desarrollado en 1979 por Ron Rivest, Adi Shamir, y Leonard Addleman. El nombre de este algoritmo viene descrito por las iniciales de sus apellidos. La característica principal del algoritmo sería la dificultad de cómputo a la hora de factorizar 2 números primos de gran tamaño.

2.3.2.2 Generación de claves en RSA

Lo primero que vamos a abordar sería la creación de esas claves. Para ello elegiremos aleatoriamente esos 2 números primos de gran tamaño y calcularemos el producto

$$n = p \cdot q$$

Guardaremos p y q en secreto además de $\phi(n)$

La función ϕ hace referencia a la función de Euler. Su definición seria la siguiente:

$$\phi(n) = card(i)\{i \in N \qquad 1 \leq i < n \qquad \text{mcd}(i,n) = 1\}$$

Después de esto elegiremos un número natural e tal que:

$$0 < e < \phi(n) \qquad \text{primo con } \phi(n)$$

O lo que sería lo mismo:

$$\text{mcd}(e, \phi(n)) = 1$$

Normalmente este número es 65537 ya que es considerado un valor lo suficientemente grande como para evitar ataques. Esto sería necesario para asegurarnos de que el número elegido

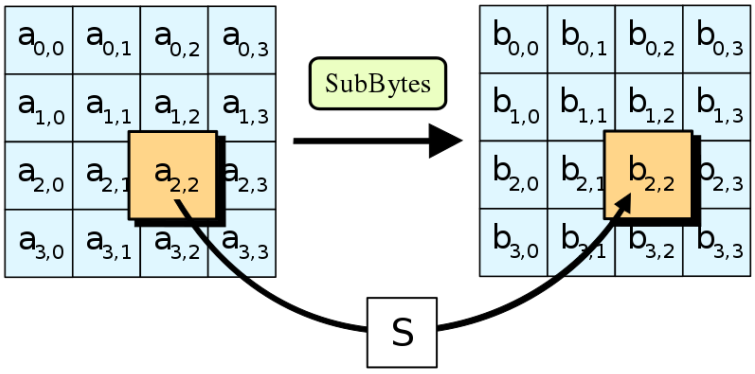

Figura 2.6: AES - SubBytes.

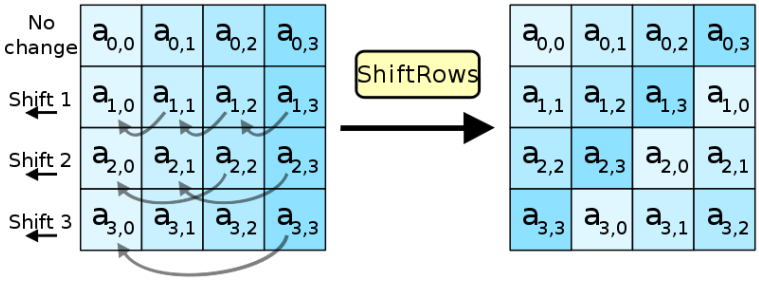

Figura 2.7: AES - ShiftRows.

e tiene inverso multiplicativo y de esta manera existiría su clave privada inversa.

$$\text{mcd}(d, \phi(n)) = 1$$

Las funciones de cifrar y descifrar serian las siguientes:

Para la clave pública:

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

Para la clave privada:

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

2.4 Comunicaciones Seguras

Aparte de cifrar la información sensible que almacenaremos en el servidor también tendremos que cifrar las comunicaciones. Para ello usaremos las siguientes técnicas:

1. Usaremos HTTPS/TLS para segurizar el canal.

2. Usaremos la tokenización de nuestras peticiones para de esta manera saber en todo momento que el usuario es quien dice ser.

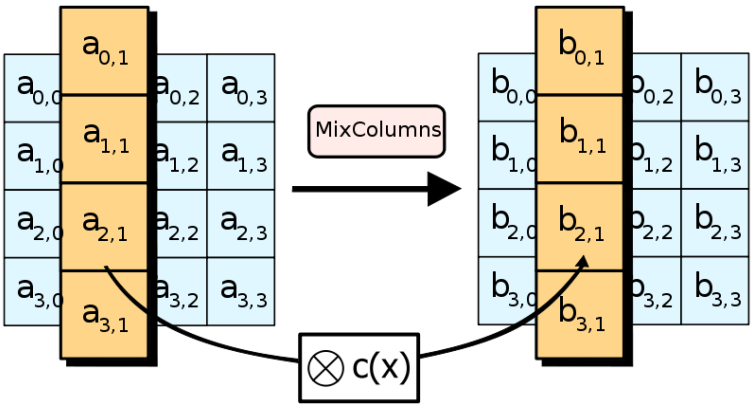

Figura 2.8: AES - MixColumns.

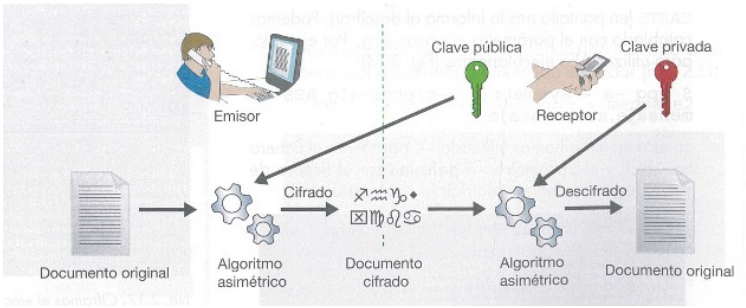

Figura 2.9: Cifrado Asimétrico.

2.4.1 HTTPS/TLS

La comunicación de nuestra aplicación la haremos mediante mensajes por la red. Podríamos usar HTTP pero estos mensajes serian totalmente visibles para cualquiera que los intercepte, no habría ningún tipo de seguridad. Es por ello que usaremos HTTPS/TLS.

El protocolo TLS es una evolución de su antecesor SSL. Este es el que evoluciona a HTTP para ser HTTPS (S de secure, seguro).

El protocolo SSL usa un cifrado híbrido ya que usaremos tanto cifrado simétrico como asimétrico. El cifrado asimétrico para la negociación de los algoritmos y claves de la comunicación y para el canal usaremos un cifrado simétrico.
En este proyecto se ha implementado HTTPS/TLS con el algoritmo RSA y su certificado.

2.4.2 Certificado Digital

El uso de algoritmos de cifrado en una comunicación no es suficiente para darnos seguridad en una comunicación. El motivo viene dado por la posible suplantación de claves por parte de un atacante en la comunicación. Este podría hacerse pasar por uno de ellos o ambos para obtener las claves de descifrado.

Es por ello que es necesario contar con un tercero que verifique la autenticidad de los actores en la comunicación y de este modo darnos la seguridad de que es quien dice ser y que no se han alterado las claves.

El certificado lo crearía una Autoridad Certificadora. Suponiendo que se quiere certificar al servidor este sería creado de la siguiente manera:

$$S \qquad E_{K_S} D_{k_S}$$

$$AC \qquad E_{K_{AC}} D_{k_{AC}}$$

$$\text{Certificado de } A \qquad c_A = D_{k_{AC}}(K_A)$$

$$\text{Autenticación de } A \qquad E_{K_{AC}}(c_A) = K_A$$

De esta manera solo necesitaríamos la clave pública de la entidad certificadora para verificar que realmente somos quienes decimos ser.

2.4.3 Implementación de HTTPS/TLS

Lo primero será iniciar la comunicación y negociación de claves o Handshake (Ver **figura 2.10**). Constará de los siguientes pasos:

- El cliente solicitará abrir un canal seguro al servidor.
- El servidor le responderá la petición otorgándole su certificado RSA y clave pública.
- Después de verificar la autenticidad del certificado, el cliente creara la clave con cifrado simétrico, como AES, que se usara para la comunicación segura y cifra esta con la clave pública del servidor. Por último la envía al servidor.

Después de esto se usará la clave AES creada para la comunicación simétrica entre ambos actores de la comunicación. La ventaja de usar una clave de cifrado simétrica es la velocidad de cifrado y descifrado.

Figura 2.10: Handshake de TLS.

2.4.4 Tokens y uso en el proyecto

Los tokens funcionarán como control de validez en todas las peticiones que se le hagan al servidor por parte del cliente salvo el *login*.

Este servirá para verificar que las peticiones las está haciendo el usuario previamente logeado y no se está haciendo pasar por otro (por ejemplo con más privilegios). Para este apartado se ha implementado la librería JWT que basa su autenticación en el algoritmo RSA.

La implementación del token irá en la cabecera de cada mensaje http y contará con los siguientes apartados.

- Header: Indicaremos el algoritmo para la firma y que tipo de token es.
- Payload: Indicaremos la información referente a quien se le está otorgando ese token, en nuestro proyecto lo usaremos para indicar el DNI del usuario, expiración del token y su rol.
- Verify Signature: Firma de dicho token. Usaremos el algoritmo de cifrado para firmar el token. Esta consta de los 3 componentes: **header**, **payload** y el **secreto**, el cual usaremos en el cifrado.

Toda la información del token siempre viajará codificada en base 64 pero no por ello está cifrado. Por ello podríamos pensar que remplazar la información sería algo sencillo. Es por esto que verificar la firma por parte del servidor sería fundamental para dar validez al token, en nuestro proyecto para cada petición que se le haga al servidor.

2.5 Función Hash

El algoritmo Hash no es propiamente un algoritmo criptográfico, ya que carece de clave. Su función principal es la de crear una huella digital de un bloque de datos y este será de una longitud fija por lo que si es mayor del preestablecido funcionará como compresor de dicha información.

Las funciones Hash cumplen con las siguientes características:

- Unidireccionalidad: Debe ser computacionalmente imposible encontrar un mensaje M de un resumen $h(M)$
- Compresión: Dado un mensaje M mayor que el tamaño prefijado de la función h el resultado de $h(M)$ no sobrepasara el tamaño prefijado. Por el contrario si el mensaje M tiene menor tamaño que le prefijado este siempre aumentará el tamaño hasta alcanzar el prefijado.
- Bajo Costo: Debe ser una función rápida.
- Difusión de bits: Si cambiáramos un solo bit del resumen resultante este debería cambiar aproximadamente en la mitad de sus bits.

- Resistencia débil a colisiones: Se debe cumplir que no haya el mismo resumen posible para dos mensajes diferente.

Las funciones hash pueden ser usadas junto a algoritmos de cifrado como es el caso de AES. En este proyecto se usará para cifrar la contraseña de los usuarios del sistema.

2.6 Contabilidad Básica

La aplicación que se va a desarrollar en el proyecto está enfocada a los fundamentos básicos para la gestión de apuntes contables de empresas. A continuación realizaremos un pequeño análisis de los conceptos básicos que se usaran en ella. Los siguientes puntos serian los más relevantes:

- Activo: Se define como el conjunto de bienes, derechos de una persona/empresa.
  - Activo circulante: Serán aquellos bienes y derechos que sean adquiridos por la persona/empresa por menos de un año.
  - Activo no circulante: Serán aquellos bienes y derechos que sean adquiridos por la persona/empresa por más de un año.
- Pasivo: Se define como el conjunto de obligaciones que tiene una persona/empresa.
  - Pasivo circulante: Serán aquellas obligaciones de la persona/empresa que vayan a ser saldadas en menos de un año.
  - Pasivo no circulante: Serán aquellas obligaciones de la persona/empresa que vayan a ser saldadas a más de un año vista.
- Patrimonio: Se define como la diferencia entre el activo y el pasivo de una persona/empresa.
- Cuentas contables: Se define como el conjunto de registros de las transacciones para un ente económico. Estas cuentas se registrarán en forma de asientos contables. Podrán tener un origen de débito o crédito.
- El ”debe”: Se encargará de registrar un ingreso o débito.
- El ”haber”: Se encargará de registrar gasto o crédito.
- Balance: Diferencia entre el debe y el haber.

Nuestra aplicación realizará apuntes contables en los que podremos cotejar si las cuentas contables están en balance, necesario para realizar el **cierre contable**.

2.7 Programas de contabilidad

En la actualidad existen muchas aplicaciones para realizar una contabilidad sencilla, tanto en el ámbito privativo como en el de software libre.

En este último caso podríamos hablar de FacturaScript, programa de contabilidad en la nube. En él nos presentan un programa de fácil acceso, cosa en la que nos fijaremos a la hora de realizar el proyecto.

Uno de los puntos fuertes que tiene la aplicación es la gestión en la nube de los contenidos para los cuales realizaremos el trabajo contable. Uno de los objetivos del proyecto será explorar caminos para la segurización de una aplicación que va a usar la conexión a internet como canal de comunicación.

3 Objetivos

Para este proyecto se plantearon una serie de hitos que se detallaran a continuación.

3.1 Programa de contabilidad básico

El principal objetivo es el de desarrollar una aplicación cliente/servidor en el que se implemente un sistema de contabilidad básico. El sistema será usado por 2 tipos de roles diferentes, usuario y administrador.

El usuario será el encargado de gestionar las diferentes empresas que tenga asociadas con sus ejercicios. Cada uno de estos ejercicios constarán de 2 diarios, borrador y general, los cuales nos servirán para gestionar cada asiento/apunte de ese ejercicio.

Además podremos generar listados de los diarios generales así como también de las cuentas de este. Por otra parte el administrador será el encargado de dar de alta a empresas, ejercicios contables y usuarios, además de otorgar permisos entre ellos. El objetivo de esto es tener más seguridad a la hora de gestionar empresas y otorgar permisos a estas.

3.2 Base de datos en el servidor

Será necesario el desarrollo de una base de datos para la gestión de la contabilidad de las empresas y los usuarios del sistema. Estará dividida en 2 secciones:

- Base de datos sin cifrar: No encontraremos información sensible, solamente datos básicos de las empresas y de los usuarios para el registro.
- Base de datos cifrada con clave simétrica: Tendremos toda la información referente a los ejercicios contables de cada empresa.

De esta manera podremos proteger la información guardada por posibles ataques de ciberdelincuentes.

3.3 Comunicación segura

Se pretende que toda comunicacion viaje cifrada en el canal mediante HTTPS/TLS con RSA, así como la implementación de tokens JWT con RSA.

4 Metodología

4.1 Plan de trabajo

Este se ha dividido por hitos de 15 días en los que se iba avanzando con el desarrollo progresivamente. En estos hitos se realizaba una reunión en el que se mostraba los avances, se preguntaban dudas surgidas y consejos para el desarrollo de los siguientes puntos a realizar.

Cada uno de los hitos se fue desarrollando con metodologías ágiles, las cuales están compuestas de como mínimo una semana para la implementación del hito en sí. Para todas las dudas y problemas que surgían en el desarrollo se han realizado reuniones mediante videoconferencia en el que se exponían los puntos a debatir y se discutía sobre las diferentes opciones de mejora.

El desarrollo se dividió en los siguientes hitos:

- Desarrollo de los mockups del cliente.
- Desarrollo de la primera versión de la base de datos. Sin cifrar.
- Desarrollo del cliente básico basado en los mockups ya diseñados.
- Desarrollo del servidor básico.
- Modificación de la base de datos para incluir cifrado de información sensible.
- Incluir cifrado del canal y tokens.
- Incluir firma en listados mediante certificado digital.

4.2 Planificación

Se usó Trello, ver **figura 4.1**, para tener claras las prioridades en cada hito así como para llevar una gestión de las tareas a realizar de manera intuitiva. Para ello se creó un tablero en el que seguir el desarrollo del proyecto usando metodologías ágiles como Scrum.

4.2.1 Scrum

Scrum es la metodología de trabajo en la que nos basaremos para el desarrollo ágil del software. En ella se engloban un conjunto de buenas prácticas para el desarrollo en equipo y obtener resultados de manera rápida. Una de las características es que los requisitos pueden estar poco definidos donde se prima la flexibilidad y productividad.

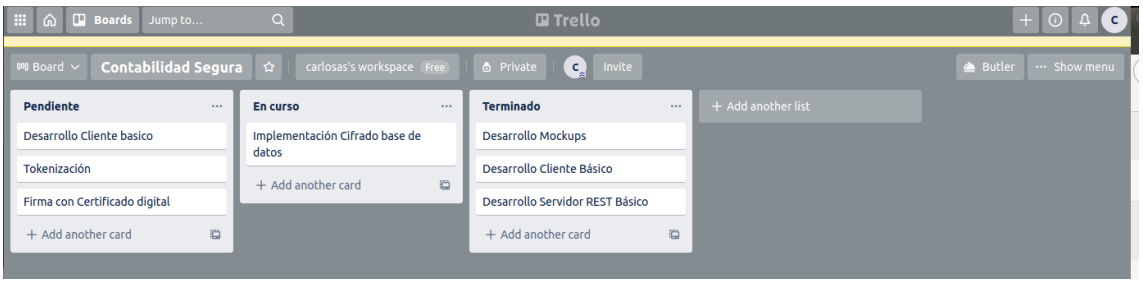

Figura 4.1: Trello.

La manera de planificar el trabajo se realizará mediante iteraciones las cuales tendrán normalmente un tiempo de 2 semanas en las que al acabar la iteración se proporcionara un resultado completo.

En este proyecto al ser un trabajo individual no se tiene la premisa de trabajo en equipo, pero si la de una productividad en la que los objetivos se establecen a corto plazo. Por ello se ha decidido optar por esta forma de organización para conseguir buenos resultados y tiempo de mejora ante adversidades de una mala planificación anterior.

4.3 Control de versiones

Es importante tener copias de seguridad del trabajo realizado. Una de las maneras más seguras de mantener nuestro trabajo es hacer uso de herramientas de control de versiones. Gracias a estas herramientas podremos tener un registro de los cambios realizados para un proyecto en el que podremos recuperar versiones modificadas de los archivos pertenecientes a este.

Podremos hacer uso de la herramienta de manera local como centralizada, como puede ser el de un proyecto en el que desarrollen software varias personas. Para este proyecto usaremos Git, herramienta de control de versiones referente en la industria.

5 Desarrollo

El objetivo principal de este proyecto es realizar la estructura básica de una aplicación REST segura. Para ello es importante realizar un análisis previo a los objetivos anteriormente descritos. A partir de estos objetivos se desarrollará la arquitectura base de nuestra aplicación.

5.1 Arquitectura del sistema

El proyecto está separado en 2 partes diferenciadas: cliente y servidor (ver **figura 5.1**). Estos se comunicarán para dar soporte al usuario que lo esté usando en ese momento, bien sea el administrador del sistema o un cliente. Para este cometido se realizará la comunicación mediante llamadas REST al servidor y este dará soporte a cada una de las llamadas.

- Aplicación del servidor
  - Estructura API/REST
  - Base de datos
- Aplicación del cliente
  - Frontend de gestión
  - Firma listados con certificado digital
- Segurización del canal

Podríamos enfatizar la importancia de estas llamadas y de emplear mecanismos de seguridad en cada una de estas peticiones al servidor.

5.2 Tecnologías usadas

Para el desarrollo del proyecto se han elegido los lenguajes de programación y gestor de base de datos que más se han acercado a la visión de este. A continuación se verán con más detalle.

5.2.1 Go

Tanto para la aplicación del cliente como la del servidor (API/REST) usaremos Go.

Este lenguaje es desarrollado por Google principalmente enfocado a ser un lenguaje de fácil uso y enfocado en API/REST. Es por ello que ha sido el lenguaje elegido para el *backend* del

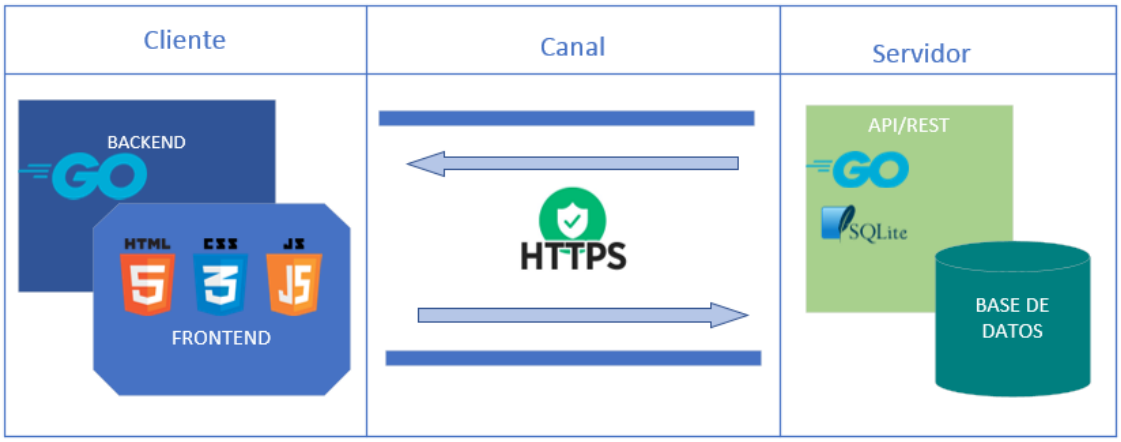

Figura 5.1: Diagrama del proyecto.

proyecto.

Por parte del cliente usaremos principalmente la librería *lorca* para establecer una conexión entre el *backend* y el *frontend* desarrollado en HTML/Javacript. Se crearán las estructuras necesarias para llevar a cabo la lógica de negocio y realizar peticiones REST al servidor.

Por parte del servidor se desarrollará la API/REST para dar soporte al cliente y se usarán las técnicas de cifrado y segurización implementando diferentes arquitecturas y algoritmos como es el caso de HTTPS/TLS o JWT.

5.2.2 HTML/ JavaScript / JQuery

El apartado visual de la aplicación será desarrollado en HTML / Javascript / JQuery.

Cada sección de la aplicación estará dividida en las diferentes páginas y se usará de la actualización dinámica de estas para evitar las recargas innecesarias de la aplicación. Solo usaremos estas tecnologías para representar el entorno visual y nunca para realizar llamadas al servidor. De esta manera nos aseguramos de que todas las peticiones cumplan con los parámetros seguros implementados en *backend*. Para ayudarnos en una fácil implementación de aspectos gráficos usaremos el *framework* Bootstrap, añadiendo estilos css complementarios para casos concretos como colores o pantallas de carga.

5.2.3 Sqlite3

Al ser un proyecto en el que la base de datos no va a contar con gran cantidad de tamaño ni por tablas y ni por ser un trabajo profesional, sino más bien académico se ha optado por trabajar con *sqlite3*.

Una de las ventajas que plantea el trabajar con este gestor de base de datos es que el contenido se resume a un único archivo. Esta característica nos da la posibilidad de decidir para

cada una de estas bases de datos individuales el nivel de protección y como haremos el cifrado.

Usaremos tanto base de datos en claro, puramente *sqlite3*, como su versión de cifrado simétrico con AES llamada *sqlcipher*. Estas bases de datos serán las que usaremos para guardar los ejercicios de cada una de las empresas.

5.3 Aplicación del servidor

Podríamos dividir el servidor en las siguientes secciones:

- Estructura API/REST
- Base de datos

5.3.1 Estructura API/REST

La gestión de los datos, manipulación y conservación, se realizará en la base de datos del servidor. Para desempeñar esta tarea se realizarán llamadas desde los diferentes clientes que usen la aplicación. Por ello la tarea principal del servidor es atender a cada una de las peticiones.

Se ha optado por una estructura de modelo vista controlador (ver **figura 5.2**) para separar el acceso a la base de datos de la lógica interna de cada una de las llamadas. Es por ello que cada una de las posibles peticiones a la base de datos tendrá su controlador individual y por otra parte su modelo en el que se realizarán estas llamadas para gestionar la petición.

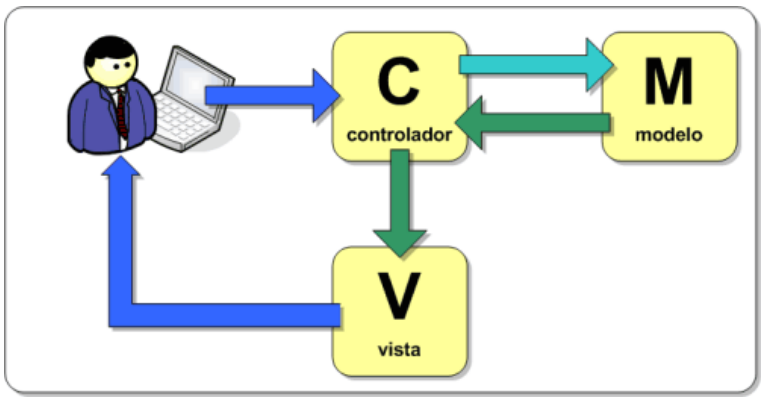

Figura 5.2: Modelo/Vista/Controlador.

Las peticiones que se podrán realizar vendrán dadas por las rutas preestablecidas desde el servidor que responderán a las necesidades de los clientes por obtener información (GET), insertar datos en la base de datos (POST, PUT), o eliminarla (DELETE).

Para asegurarnos de otorgar la petición al cliente si tiene los permisos se realizará una comprobación mediante *tokens*.

Para la generación de los *tokens* el servidor usará criptografía asimétrica gracias a JWT y RSA. Cuando el usuario realice el *login* correctamente el servidor le entregara un token firmado con su clave pública.

Como podemos ver en la **figura 5.3** los *tokens* JWT constan de 3 partes; *header*, *payload* y firma. En el *header* se especificará el algoritmo de cifra del token y en el *payload* tendremos la información de este, en nuestro caso será la información de a quien se le ha otorgado, el rol y el tiempo de extinción del mismo. En el último campo tendremos la firma que será lo que usemos para comprobar la validación de cada llamada REST.

Figura 5.3: Ejemplo JWT.

Para la creación y comprobación de estos tokens usaremos la librería *jwt-go* en la que se nos darán las herramientas necesarias para el uso de *tokens* JWT.

```
1 //Middleware antes de procesar las peticiones.
2 func ValidateTokenMiddleware(next http.HandlerFunc) http.HandlerFunc {
3     return func(w http.ResponseWriter, r *http.Request) {
4         if autentication.ValidateJWT(w, r) {
5             next.ServeHTTP(w, r)
6             return
7         }
8         utils.Response(w, false, "invalid Token")
9     }
```

```
}
```

```
//Validación JWT
func ValidateJWT(w http.ResponseWriter, r *http.Request) bool {
    token, err := request.ParseFromRequestWithClaims(r, request.OAuth2Extractor, &models.Claim
        ↪ {}, func(token *jwt.Token) (interface{}, error) {
        return publicKey, nil
    })
    var isValid = false
    if err != nil {
        switch err.(type) {
        case *jwt.ValidationError:
            vErr := err.(*jwt.ValidationError)
            switch vErr.Errors {
            case jwt.ValidationErrorExpired:
                fmt.Println("token expired")
                return isValid
            case jwt.ValidationErrorSignatureInvalid:
                fmt.Println("signature invalid")
                return isValid
            default:
                fmt.Println("the token is invalid")
                return isValid
            }

        default:
            fmt.Println("the token is invalid")
            return isValid
        }
    }
    if token.Valid {
        isValid = true
    } else {
        fmt.Println("Your token is NOT valid")
    }

    return isValid
}
```

5.3.1.1 Ejemplo Ruta/Controlador/Modelo

A continuación se expondrá un ejemplo de la arquitectura del servidor basándonos en un recurso dado, en este caso la obtención de conceptos contables.

En el servidor daremos de alta las rutas en las que se podrán realizar las peticiones REST de conceptos contables. El primer paso será comprobar la validez del token y después de esto procesar la llamada REST.

```
var RegisterConceptRoutes = func(router *mux.Router) {
    router.HandleFunc("/concept/{company}", c.ValidateTokenMiddleware(c.CreateConcept)).
        ↪ Methods("POST")
```

```
    router.HandleFunc("/concept_get/{company}", c.ValidateTokenMiddleware(c.GetConcept)).↩
        ↪ Methods("POST")
    router.HandleFunc("/concept_id/{company}", c.ValidateTokenMiddleware(c.GetConceptByID)).↩
        ↪ Methods("POST")
    router.HandleFunc("/concept/{company}", c.ValidateTokenMiddleware(c.UpdateConcept)).↩
        ↪ Methods("PUT")
    router.HandleFunc("/concept/{company}", c.ValidateTokenMiddleware(c.DeleteConcept)).↩
        ↪ Methods("DELETE")
    router.HandleFunc("/concept_struct/", c.ValidateTokenMiddleware(c.GetConceptStruct)).↩
        ↪ Methods("GET")
}
```

El siguiente paso seria, en función de la petición, que el controlador realizara las operaciones de lógica de negocio, entre otras obtener los datos de la llamada y comunicarse con el modelo para su procesamiento. Por último se encargaría de devolver esa información al cliente.

```
func GetConceptByID(w http.ResponseWriter, r *http.Request) {

    var concept models.ConceptKey // Tomando el cuerpo de la petición, en formato JSON, y

    body, err := ioutil.ReadAll(r.Body)
    if err != nil {
        fmt.Println(err)
    }
    if err := json.Unmarshal(body, &concept); err != nil {
        panic(err)
    }

    companydb, err := strconv.Atoi(mux.Vars(r)["company"])

    errAttach := models.AttachDatabase(concept.Key, companydb)
    if errAttach != nil {
        panic(errAttach)
    }

    c, err := models.GetConceptByID(concept.Structure.ID)

    errAttach = models.DetachDatabase()
    if errAttach != nil {
        panic(errAttach)
    }

    if err != nil {
        utils.Response(w, false, err.Error())
        return
    }
    w.WriteHeader(http.StatusOK)
    w.Header().Set("Content-Type", "application/json")
    utils.ResponseData(w, true, "Query Successfully", c)
}
```

En la llamada al modelo se establecerá la comunicación con la base de datos para la obtención de la petición por parte del cliente.

```
func GetConceptByID(id int) (Concept, error) {
    c := Concept{}

    db := GetConnection()
    var q string

    q = `SELECT id, title
            FROM concept
            WHERE id = ?`

    row := db.QueryRow(q, id)

    err := row.Scan(
        &c.ID,
        &c.Title,
    )

    if err != nil {
        return c, err
    }

    return c, nil
}
```

5.3.2 Base de datos

La base de datos la usaremos para guardar toda la información referente a las gestiones contables y a los usuarios. Para ceñirnos al cifrado de la información sensible se ha optado por separar la base de datos en 2 secciones (ver **figura 5.4**).

Por una parte tendremos la gestión de los usuarios/empresas y sus permisos a la que llamaremos **base**. Esta será la base de datos que no cifraremos. Los permisos de estos usuarios a las empresas que van a gestionar serán los que les otorgarán el acceso a sus correspondientes bases de datos cifradas (ver **figura 5.5**)

Por otra parte estará la información contable de empresas o **diarios** que estará cifrada y separada de la **base** para cada una de las empresas dadas de alta.

5.3.2.1 Empresas, usuarios y permisos

Las **empresas** contarán para identificarlas con un código autogenerado, su nombre y su NIF.

Lo **usuarios** podrán identificarse con 2 roles, **cliente** y **administrador**.

El **cliente** podrá tener acceso a una o más empresas. Esto le permitirá realizar las operaciones de gestión para los ejercicios que estén dados de alta en el sistema para esas empresas.

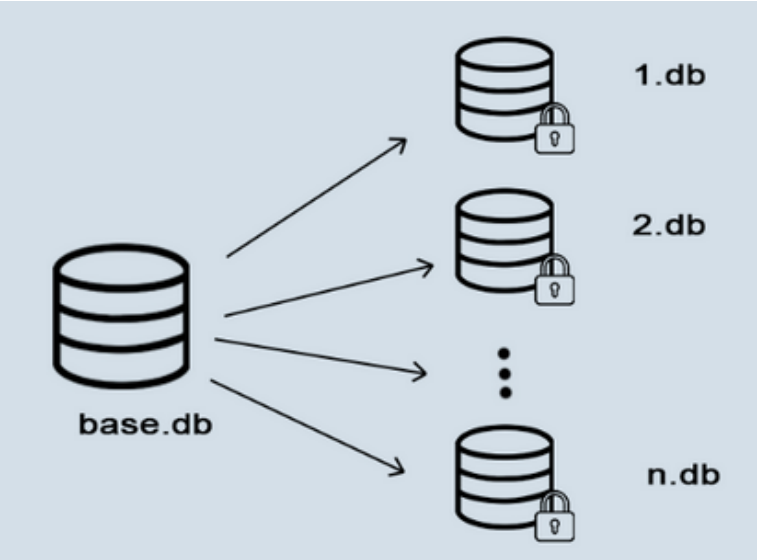

Figura 5.4: Esquema base de datos.

El **administrador** tendrá la capacidad de dar de alta las empresas, los ejercicios contables y los usuarios en el sistema, ya sean clientes u otros administradores.

5.3.2.2 Diarios

Las bases de datos cifradas de cada empresa guardarán principalmente los diarios contables. Su diagrama es el siguiente:

Podremos gestionar 1 o muchas empresas y estas a su vez podrán tener uno o muchos **ejercicios contables**. Cada uno de estos ejercicios siempre tendrán un **diario general** y un **diario borrador**. Cada uno de estos diarios tendrán **asientos contables**. Los asientos contables contarán con uno o más **apuntes contables** (ver **figura 5.6**).

5.4 Aplicación del cliente

La aplicación del cliente esta divida principalmente en 2 secciones, *frontend* y *backend*. La lógica siempre la trabajará el *backend* asumiendo por tanto la parte visual únicamente al *frontend*.

5.4.1 Estructura backend

Las tareas a realizar por el *backend* serán las siguientes:

- Comunicación con el *frontend*
- Peticiones API/REST por parte de cliente.

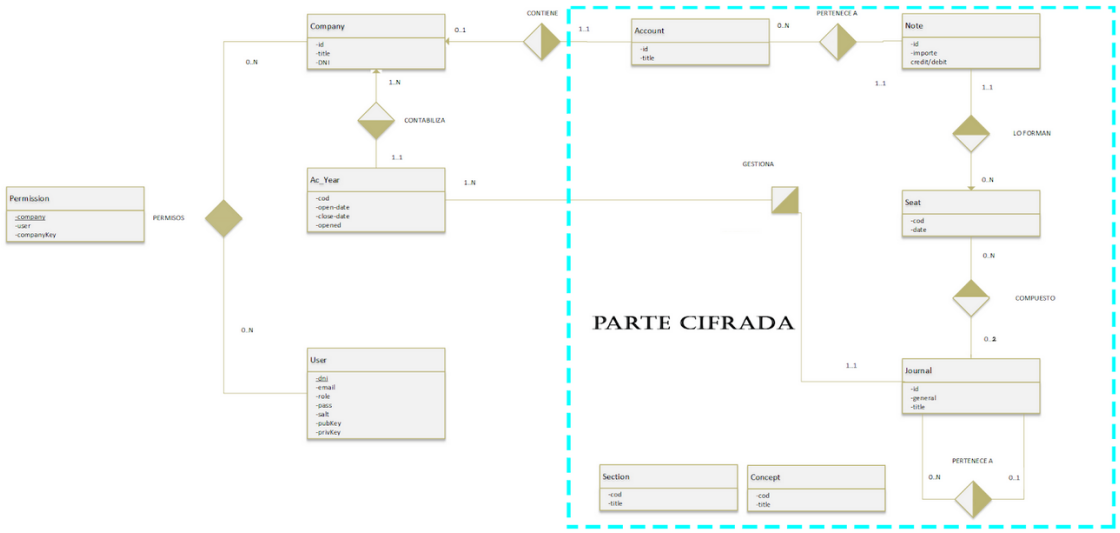

Figura 5.5: Diagrama de la base de datos.

- Cifrado y descifrado por parte del cliente.

5.4.1.1 Comunicación con el frontend

Como se ha mencionado anteriormente se hará uso de la librería *lorca*. Gracias a *lorca* podremos crear un entorno gráfico para nuestro cliente apoyándonos en el explorador Chrome. Esto nos facilitará el desarrollo de vistas usando HTML.

Podremos diferenciar varias secciones para el uso de esta librería.

1. Iniciar *lorca* seleccionando los parámetros necesarios para nuestras especificaciones como el S.O. en el que funcionará la aplicación o el tamaño de pantalla seleccionado.

2. Crear los enlaces con las funciones que se podrán llamar desde las páginas HTML/JS mediante la función *Bind*.

```
func main() {
    c := models.Client{}
    args := []string{}

    if runtime.GOOS == "linux" {
        args = append(args, "--class=Lorca")
    }

    ui, err := lorca.New("", "", 1200, 700, args...)
    if err != nil {
        panic(err)
    }
    defer ui.Close()
    myui := MyUI{ui, c}

```

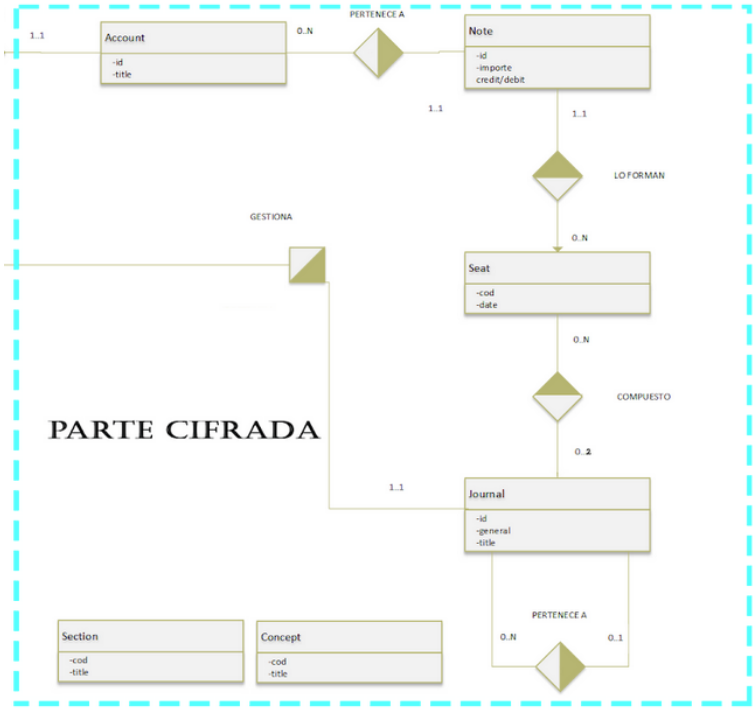

Figura 5.6: Diagrama de Diarios.

```
16    myui.chargeView("./www/index.html")
17    ui.Bind("chargeView", myui.chargeView)
18    ui.Bind("chargeView2", myui.chargeView2)
19
20    ui.Bind("SignIn", myui.c.SignIn)
21    ui.Bind("SignUp", myui.c.SignUp)
22    .
23    .
24    .
25    sigc := make(chan os.Signal)
26    signal.Notify(sigc, os.Interrupt)
27    select {
28    case <−sigc:
29    case <−ui.Done():
30    }
```

3. Realizar la carga de plantillas (*Templates*) en Go mediante la función *Load* con la posibilidad de enviar parámetros desde el *backend* a nuestra vista y así establecer comunicación directa de datos mediante la navegación.

```
1 func (myui *MyUI) chargeViewWithParam(filePath string, id int) {
2
3     tmpl, err := template.ParseFiles(filePath)
4     if err != nil {
5         panic(err)
6     }
```

```
7
8     buff := bytes.Buffer{}
9
10    err = tmpl.Execute(&buff, id)
11    if err != nil {
12        panic(err)
13    }
14    myui.ui.Load("data:text/html," + url.PathEscape(string(buff.Bytes())))
15}
```

.

5.4.1.2 Peticiones API/REST por parte de cliente

Se realizarán peticiones al servidor con funciones globales en el que podremos diferenciar peticiones GET, POST, PUT y DELETE. Estas llamadas se diferenciarán de sí se hacen a la base de datos cifrada de cada una de las empresas (por ejemplo petición GET para obtener los apuntes contables de una empresa especifica) o se está haciendo a la base (GET para obtener los usuarios del sistema). Las peticiones a las bases de datos específicas necesitarán de la clave para poder acceder a esta por lo que antes de todo se tendrá que realizar la comprobación de sí el usuario que está pidiendo la información tiene acceso a esa empresa.

```
1//GetStructure make a request to get the generic row
2func (c *Client) GetStructure(table string) (interface{}, error) {
3
4    var response = utils.RespData{}
5
6    req, err := http.NewRequest("GET", "https://localhost:9043/"+table+"_struct/", nil)
7    req.Header.Add("Authorization", "Bearer "+c.token)
8
9    if err != nil {
10       log.Fatal(err)
11   }
12   res, err := c.httpclient.Do(req)
13
14   if err != nil {
15       log.Fatal(err)
16   }
17   defer res.Body.Close()
18   //Unmarshal the response to a resp struct
19   body, err := ioutil.ReadAll(res.Body)
20
21   json.Unmarshal(body, &response)
22
23   return response, nil
24}
```

5.4.2 Estructura frontend

La estructura de la visualización de cliente podría resumirse en los siguientes puntos:

- Páginas html de funcionalidades.

- Funciones Javascript/JQuery para procesamiento de datos.

5.4.2.1 Páginas html de funcionalidades

Estas han sido diseñadas con la simplicidad de ser lo más fáciles de entender por el usuario. La implementación de sus formularios nos darán paso a poder realizar las peticiones REST de una manera cómoda y sencilla.

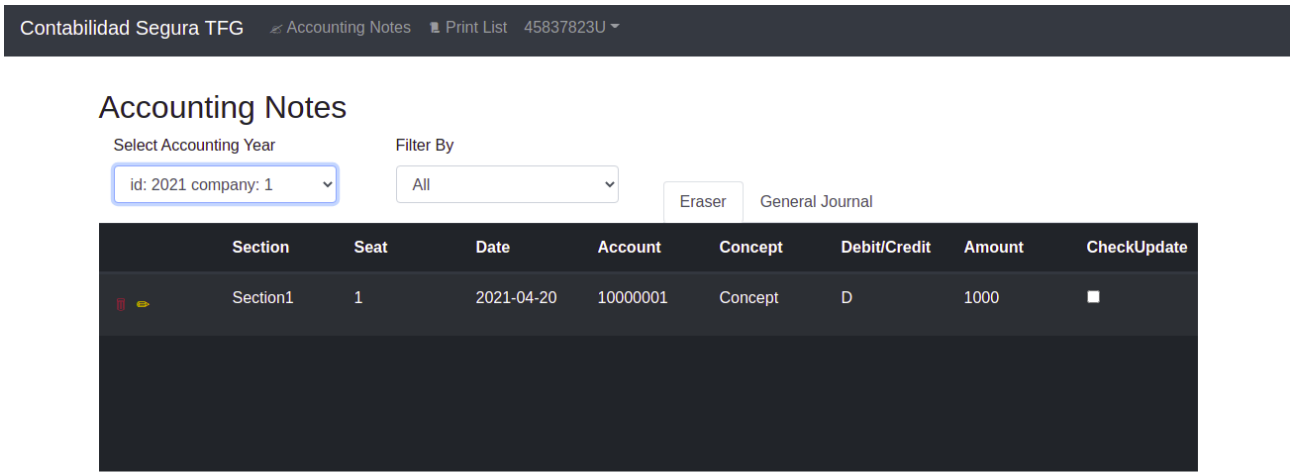

Figura 5.7: Formulario apuntes contables.

5.4.2.2 Funciones Javascript/JQuery para procesamiento de datos

La mayoría de las funcionalidades del frontend han sido programadas en un único documento js. En él se encontrarán las funcionalidades moduladas de manera que puedan realizar las llamadas a las diferentes tablas del servidor de manera dinámica. También podremos encontrar funciones de recarga de pantalla dinámica y búsqueda de elementos en el árbol DOM para aumentar en eficiencia y rapidez la aplicación.

```
async function sendInsertRow(obj,table){
    try{
        if(table =="user"){
            response = await SignUp(obj.dni,obj.name,obj.email,obj.role,obj.pass);// Call Go function
            if(!response.Ok){
                alert(response.Msg);
            }
        }else{
            response = await CreateRow(obj,table,attach);// Call Go function
            if(!response.Ok){
                alert(response.Msg);
            }
        }
    }catch(error){
        alert(error)
```

```
    }
    if(IS_AUTOINCREMENT) {
        obj.id = response.Msg;
    }
    return obj;
}
```

5.4.3 Firma de listados

Una de las funcionalidades, aparte de la gestión contable, será la de generar listados de los diarios generales y de cuentas. Estos podrán generarse gracias a peticiones REST al servidor. Para más seguridad se requiere introducir de un certificado digital por parte del cliente para firmar a su nombre los distintos archivos generados.

Para realizar esta funcionalidad nos ayudaremos de la librería *pdf-simple-sign* y disponer la copia del certificado digital en formato p12.

```
func signPDF(filename string) error {

    inputPath := filename + ".nosign"
    outputPath := filename + ".pdf"

    doc, err := ioutil.ReadFile(inputPath)
    if err != nil {
        return err
    }
    p12, err := ioutil.ReadFile("./signature/key-pair.p12")
    if err != nil {
        return err
    }
    key, certs, err := pdf.PKCS12Parse(p12, "")
    if err != nil {
        return err
    }
    if doc, err = pdf.Sign(doc, key, certs, 4096); err != nil {
        return err
    }
    if err = ioutil.WriteFile(outputPath, doc, 0666); err != nil {
        return err
    }
    return nil
}
```

5.5 Técnicas criptográficas

A lo largo del proyecto se han usado técnicas criptográficas tanto en el cliente como en el servidor. Las claves usadas para la realización de dichas técnicas son las siguientes:

- *password*: Contraseña en texto claro introducida en el cliente.

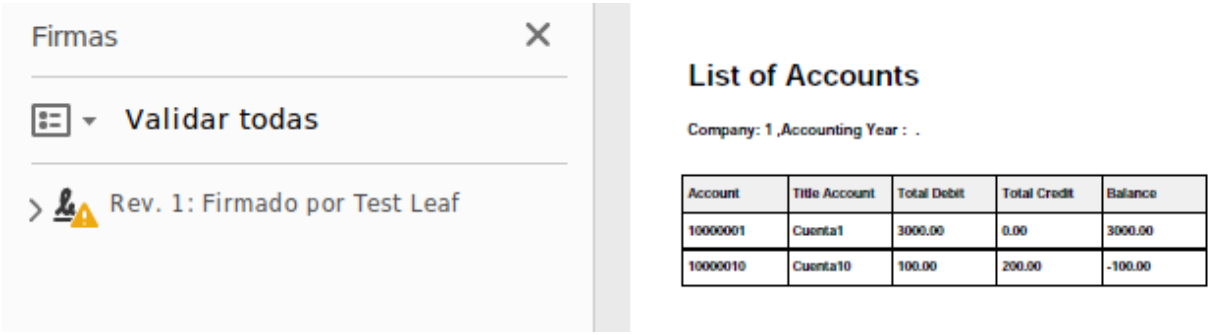

Figura 5.8: Ejemplo listado firmado.

- *passwordLogin*: Contraseña que usaremos para realizar *login* en el servidor. La creamos en el cliente a partir de la primera mitad de realizar *hash* a *password*.
- *passwordCipher*: Contraseña que usaremos para realizar cifrado simétrico desde el cliente. La creamos en el cliente a partir de la segunda mitad de realizar *hash* a *password*.
- *salt*: Semilla que usaremos junto a *passwordLogin* antes de guardarla en el servidor generando así *passwordHashed*.
- *passwordHashed*: Será la contraseña que guardaremos en el base de datos del servidor al hacer un registro, o comprobaremos en caso de *login*, de un usuario.
- *publicKey*: Clave pública (algoritmo RSA) de un usuario. Se creará al dar de alta un usuario desde el cliente. La usaremos para cifrar datos que se guardaran en el servidor, como es el caso de *companyKey* o *dataKey*.
- *privateKey*: Clave privada (algoritmo RSA) de un usuario. Se creará al dar de alta un usuario desde el cliente. La usaremos para poder descifrar lo cifrado por la clave pública publicKey de dicho usuario.
- *companyKey*: Las claves de cifrado simétrico con la que se cifrará la base de datos específica de cada empresa. Esta se creará al dar de alta una empresa. La clave se guardará en el servidor a nombre del usuario al que pertenezca su creación (el administrador) y posteriormente a los usuarios que se les otorgue los permisos de la misma. La clave, por lo tanto, se cifrará con la clave pública *publicKey* de cada usuario que tenga permiso para acceder a la empresa.
- *dataKey*: Claves de cifrado simétrico con la que se cifrará información sensible de cada una de las empresas. Esta se creará al dar de alta una empresa. La clave se guardará en el servidor a nombre del usuario al que pertenezca su creación (el administrador) y posteriormente a los usuarios que se les otorgue los permisos de la misma. La clave por lo tanto se cifrará con la clave pública *publicKey* de cada usuario que tenga permiso para acceder a la empresa.
- *publicKeyServer*: Clave pública de cifrado asimétrico. Esta clave pertenece al servidor. Esta clave se usará para cifrar información que gestionará el servidor.

Atributo	Descripción
ID	Código de la empresa.
title	Título de la empresa.
NIF	NIF de la empresa

Tabla 5.1: Tabla company

- *privateKeyServer*: Clave privada de cifrado asimétrico. Esta clave pertenece al servidor. La usará el servidor para descrifrar criptogramas cifrados con publicKeyServer.

En el desarrollo de la aplicación se han aplicado técnicas criptográficas para, en la medida de lo posible, ocultar los datos de posibles ataques de ciberdelincuentes.

Como se ha comentado anteriormente la base de datos estará dividida en **base** (o sin cifrar) y **diarios** los cuales representarán a cada empresa del sistema, serán bases de datos cifradas.

5.5.1 Operaciones criptográficas de los usuarios y empresas

Los datos referentes a los **usuarios** y **empresas** serán guardados en la base de datos **base**. Realizaremos técnicas criptográficas para proteger la información referente a estos, como es el *login*, otorgar permisos o guardar claves de acceso para las bases de datos únicas de cada empresa.

Como se puede ver en la **tabla 5.1**, las **empresas** contarán para identificarlas con un código autogenerado, su nombre y su NIF.

La información perteneciente a los **usuarios** se almacena en la tabla user (ver **tabla 5.2**) en ella guardaremos los datos identificativos del usuario además de sus claves criptográficas.

Las claves criptográficas serán necesarias tanto para darse de alta en el sistema (*passwordHashed* y *salt*) como para establecer comunicación con la sección cifrada de la base de datos (*publicKey* y *privateKey*).

Para realizar el registro en el sistema necesitaremos de 2 cosas, el DNI de usuario y la contraseña (atributo *password*). Esta contraseña la generará el administrador cuando nos dé el alta en el sistema.

Como se puede ver en la **figura 5.9** el proceso de la creación/registro de esta sería el siguiente:

1. Se introduce la contraseña *password*.

Atributo	Descripción
DNI	Será la identificación de nuestro usuario.
name	Nombre del usuario.
email	Email del usuario
role	Rol del usuario, podrá ser cliente o administrador
passwordHashed	Contraseña de *login* almacenada en el servidor.
salt	Hash de seguridad con la que se ha guardado la contraseña en el servidor
publicKey	Clave publica RSA del usuario.
privateKey	Clave privada RSA del usuario cifrada con su clave de cifrado en el cliente

Tabla 5.2: Tabla User

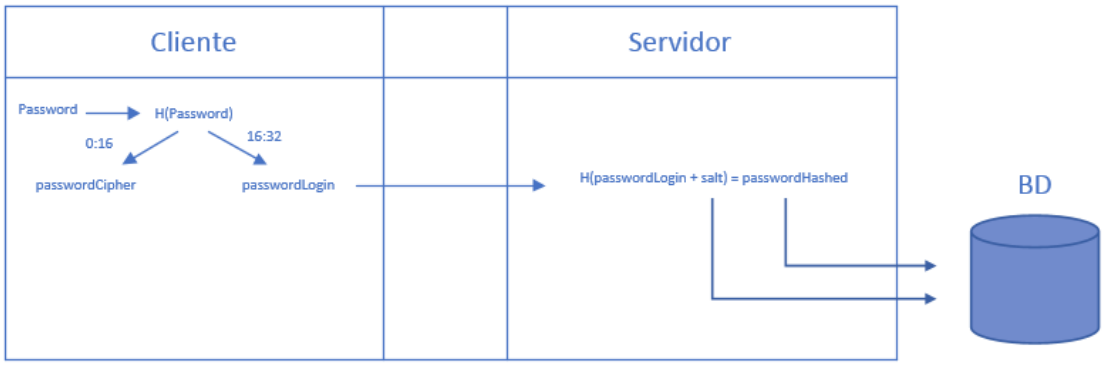

Figura 5.9: Diagrama de registro.

2. Realizaremos la función *hash* sobre *password* con el algoritmo sha256 en el que realizaremos una compresión de esta a 256 bits (32 bytes).

$$H(password)$$

3. Dividiremos la contraseña en 2 mitades. Usaremos la primera mitad de contraseña de *login* (*passwordLogin*), esta la enviaremos al servidor. La segunda la usaremos para cifrar (*passwordCipher*), nunca viajará del lado de cliente por lo que será una clave de cifrado segura.

$$passwordLogin = H(m)0:16 \qquad passwordCipher = H(m)16:32$$

4. Una vez llega al servidor se le aplicará *hash*, juntando *passwordLogin + salt* siendo esta última una semilla aleatoria.

$$passwordHashed = H(passwordLogin + salt)$$

Guardaremos tanto *salt* como el nuevo criptograma hasheado *passwordHashed* en la

base de datos.

Para realizar el *login* el procedimiento sería el siguiente(ver **figura 5.10**):

1. Realizaremos los puntos 1-4 anteriores
2. Realizaremos el *hash* con la *salt* guardada en el servidor y lo compararemos si la *passwordHashed* guardada es igual a la resultante. De ser así el usuario habrá introducido la contraseña correcta y se le otorgará su token. Si no son correctos se le comunicará el error.

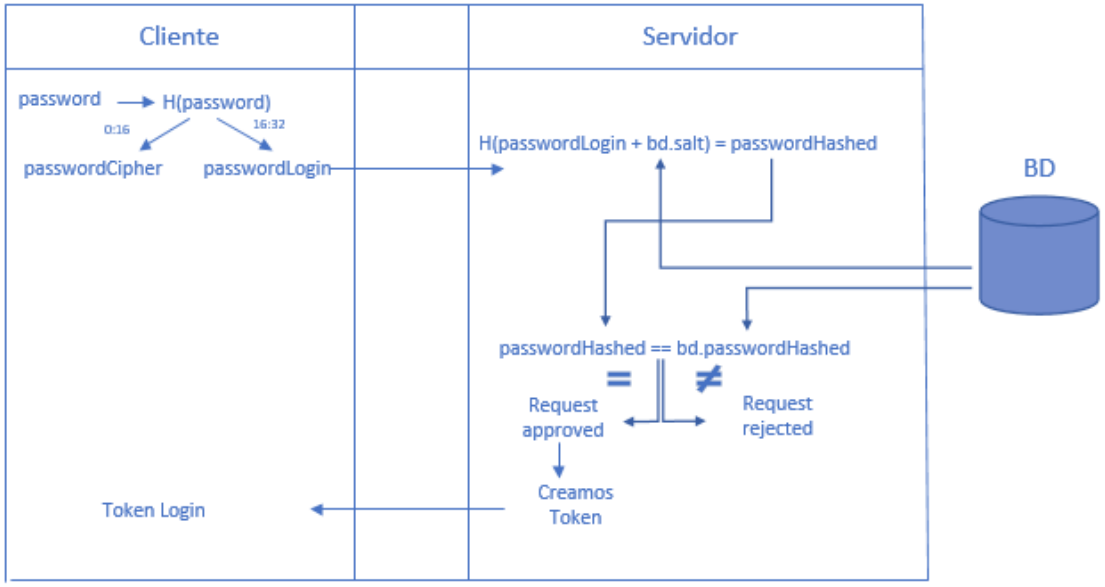

Figura 5.10: Diagrama de login.

Por otra parte para poder realizar la comunicación con las bases de datos cifradas tenemos que obtener la clave de cifrado *companyKey* para tener acceso. Para este cometido usaremos cifrado asimétrico con las claves *publicKey* y *privateKey*.

Esta será usada principalmente para la conexión con las secciones cifradas de la base de datos. Para guardar la clave privada en el servidor se cifrará con la clave de cifrado *passwordCipher*.

Para ver esto con más detalle se va a proceder a explicar la creación de estas bases de datos cifradas, como se crea la clave de cifrado *companyKey*, como la usarán los usuarios que tengan permisos y por último como usarán dicha clave para realizar sus peticiones.

La base de datos cifrada se creará cuando se realice una petición para crear una empresa nueva en el sistema. Esta acción solo la podrá realizar el administrador del sistema. Los pasos a seguir por el sistema serán los siguientes:

1. En el lado del cliente crearemos las claves *companyKey*, clave de cifrado de la nueva empresa, como *dataKey*, clave de cifrado para datos sensibles de la empresa. Esta última clave será cifrada con la clave pública del administrador y por lo tanto será el responsable de otorgar permisos a los usuarios que así considere. Estas 2 claves se enviarán al servidor.

$$companyKey \qquad E_{publicKey}(dataKey)$$

2. Haremos uso de la clave *companyKey* y posteriormente la cifraremos con la clave pública del administrador. Este paso es necesario para usar la clave sin cifrar como clave para la nueva base de datos cifrada. Guardaremos tanto *companyKey* como *dataKey* (ambas cifradas) en la **tabla 5.3** base de datos a su nombre junto al de la empresa que se acaba de crear.

Para que el administrador otorgue permisos de acceso a una empresa se realizarán las siguientes acciones (ver **figura 5.11**):

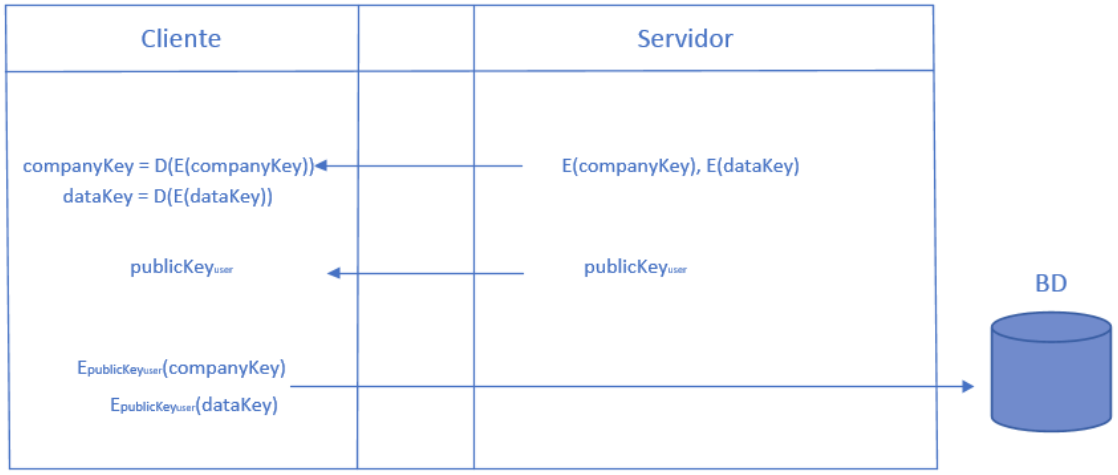

Figura 5.11: Diagrama de otorgar permisos.

1. En el lado del cliente como administrador haremos una petición para obtener *companyKey* y *dataKey*. Descifraremos ambas claves.

$$companyKey = D_{privateKey}(companyKey)$$
$$dataKey = D_{privateKey}(dataKey)$$

2. Del mismo modo obtendremos la clave pública del usuario al que se le van a otorgar permisos.

$$publicKey_{user}$$

Atributo	Descripción
company	ID de la compañia
user	DNI del usuario
companyKey	Clave de cifrado de la base de datos, cifrada con la clave pública del usuario.
dataKey	Clave de cifrado para datos sensibles de la empresa, cifrada con la clave pública del usuario.
readPermission	Indicador de permiso de lectura
writePermission	Indicador de permiso de escritura

Tabla 5.3: Tabla permission

3. Cifraremos ambas claves con la clave pública del usuario y las enviaremos al servidor.

$$E_{publicKey_{user}}(companyKey)$$

$$E_{publicKey_{user}}(dataKey)$$

4. Guardaremos tanto *companyKey* como *dataKey* en la base de datos.

Por lo tanto un usuario al que se le hayan otorgado permisos podrá realizar peticiones REST. El servidor, previa petición, se encargará de entregarle las claves de empresa, claves cifradas por su clave pública, con las que podrá realizar el enlace con la base de datos de la empresa (ver **figura 5.12**).

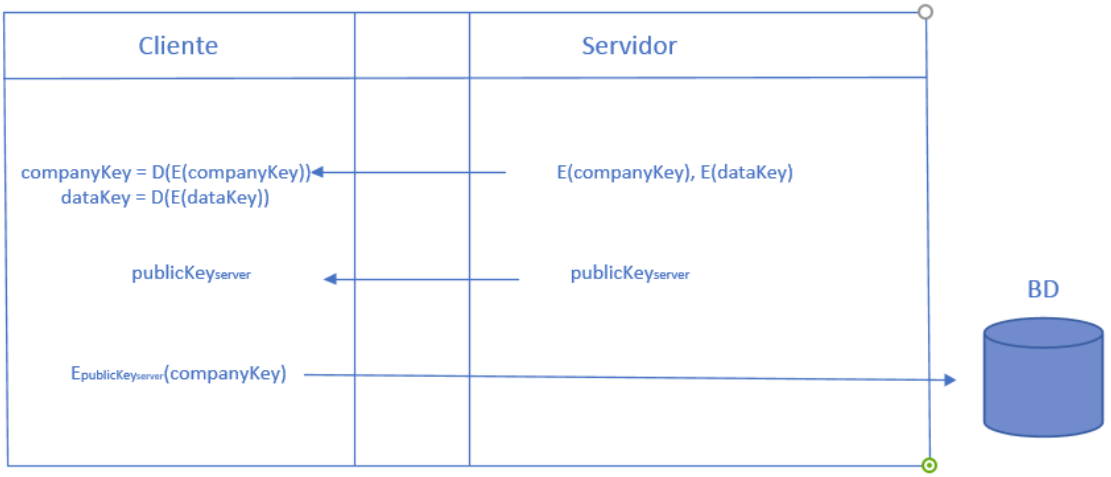

Figura 5.12: Diagrama de peticiones REST.

Esta petición ser realizará de la siguiente manera:

1. Petición para obtener la clave *E(companyKey)*

ID	ID del apunte contable
Seat	Asiento al que pertenece el apunte contable
Journal	Diario al que pertenece el apunte contable
Account	Cuenta del apunte contable
Company	Empresa de diario
Concept	Concepto del apunte
Amount	Importe del apunte (Dato segurizado)
DC	Debe o Haber (Dato segurizado)

Tabla 5.4: Tabla apuntes contables

2. Petición para obtener la clave pública del servidor para comunicaciones.

$$publicKeyServer$$

3. Descifraremos la clave usando nuestra clave privada.

$$companyKey = D_{privateKey}(E(companyKey))$$

4. Ciframos *companyKey* con la clave pública del servidor y se la enviamos para realizar enlace con la base de datos cifrada.

$$E_{publicKeyServer}(companyKey)$$

5.5.1.1 Cifrado de diarios

Para cifrar cada una de las bases de datos creadas por las empresas gestionadas haremos uso de *sqlcipher*.

Para ello usaremos la librería *go-sqlcipher* en la que usaremos el algoritmo AES256. Como hemos mencionado anteriormente, la clave que usaremos será *companyKey*, clave generada aleatoriamente en el proceso de creación de la empresa. La información de cada ejercicio contable será almacenada en la base de datos que acabamos de crear.
En el proceso de generación de la base de datos también incluiremos la creación de la clave *dataKey* que nos servirá para cifrar la información sensible de los apuntes contables. Como podemos ver en la **tabla 5.4** esta información a proteger será la referente al **importe** como **DC**.

Para poder gestionar a la información segurizada de los apuntes contables deberemos realizar las siguientes peticiones:

1. Petición para obtener la clave *E(dataKey)*

2. Descifraremos la clave usando nuestra clave privada.

$$dataKey = D_{privateKey}(E(dataKey))$$

3. Realizaremos el cifrado o descifrado simétrico sha256 con la clave *dataKey*

$$E_{dataKey}(apunteContable) \qquad D_{dataKey}(apunteContable)$$

Esta será la clave que se usará para cifrar/descifrar desde la aplicación cliente información sensible. Por ello nunca guardaremos la información en claro en el servidor y necesitaremos obtener la clave cada vez que queramos gestionar dichos datos.

Los campos elegidos para este cometido serán el importe y el carácter del apunte (debe/haber). Este proceso se realizará siempre para que en caso de obtener la clave de empresa no se puedan obtener la información sensible que hemos cifrado específicamente para ese cometido.

Los **beneficios** de esta práctica es que ante la vulneración por ciberdelincuentes y robo de información nunca podrán acceder a esos datos, ya que estarán cifrados siempre desde el cliente, sin posibilidad de acceder a la clave de descifrado en claro.

Por otra parte los **inconvenientes** de esta práctica es que todos los cálculos que se quieran hacer siempre tendrán que realizarse en el cliente, ya que en el servidor solo es información cifrada y por ello sin posibilidad de cálculo alguno (por ejemplo balance de cuentas).

```
1 //Cifrado de información sensible de apuntes contables.
2 func (c *Client) encryptDataNote(company int, structure interface{}) ([]byte, error) {
3     var response []byte
4     //First. we download the permissions and company keys and data
5 
6     permission, err := c.PermissionCompany(company)
7     if err != nil {
8         return response, err
9     }
10    encryptCompanyKey := permission.CompanyKey
11    encryptDataKey := permission.DataKey
12    //We decrypt the keys obtained with our private key
13    companyKey, err := rsa.DecryptOAEP(sha256.New(), rand.Reader, c.privKey, ↩
          ↪ encryptCompanyKey, nil)
14    if err != nil {
15        return response, err
16    }
17    dataKey, err := rsa.DecryptOAEP(sha256.New(), rand.Reader, c.privKey, encryptDataKey, nil)
18    if err != nil {
19        return response, err
20    }
21    //We obtain the public key to encrypt the channel
22    pubKeyServer, err := c.getPublicKeyOfServer()
23    if err != nil {
24        return response, err
25    }
26    //And we encrypt this with the public key of the server
27    encryptCompanyKey, err = rsa.EncryptOAEP(sha256.New(), rand.Reader, pubKeyServer, ↩
          ↪ companyKey, nil)
```

```
28      if err != nil {
29          return response, err
30      }
31      //Finally we will prepare the note data to send it to the server.
32      //We will encrypt with the Datakey using the AES algorithm the following fields of the note:
33      // − Amount
34      // − Debit/Credit
35
36      //We cast the interface with the note structure saved by the server
37      note := SeatNote{}
38      marshallNote, err := json.Marshal(structure)
39      if err != nil {
40          return response, err
41      }
42      err = json.Unmarshal(marshallNote, &note)
43      if err != nil {
44          return response, err
45      }
46
47      //We transform the fields to [] byte and then encrypt them with the data key
48      amount := utils.Float64bytes(note.Amount)
49      dc := []byte(note.DC)
50
51      amount, err = c.encrypt(amount, dataKey)
52      if err != nil {
53          return response, err
54      }
55      dc, err = c.encrypt(dc, dataKey)
56      if err != nil {
57          return response, err
58      }
59
60      //We move the data to a new structure in which we can store this information correctly to send it↩
            ↪ to the server
61      noteWithEncData := moveDataSet(note)
62      noteWithEncData.Amount = amount
63      noteWithEncData.DC = dc
64      //Finally we will create the structure with the company key
65
66      structForSend := StructAndKey{
67          Structure: noteWithEncData,
68          Key: encryptCompanyKey,
69      }
70      response, err = json.Marshal(structForSend)
71      if err != nil {
72          return response, err
73      }
74
75      return response, nil
76
77  }
```

5.5.2 Segurización del Canal

Para proteger las comunicaciones entre los dos actores de la aplicación usaremos HTTPS/TLS. Como algoritmo de cifrado usaremos RSA:

```
1
2err := http.ListenAndServeTLS(":9043", "autentication/certificate/https/server.crt", "↩
      ↪ autentication/certificate/https/server.key", nil)
3
4if err != nil {
5     panic(err)
6}
```

6 Resultados

En este capítulo mostraremos los resultados obtenidos del proyecto. La base principal del proyecto ha sido la segurización de las comunicaciones y de los datos sensibles, objetivos que se han logrado tanto en el cliente con las firmas de listados, en el servidor, con la cifra de base de datos y datos sensibles, como en el canal, gracias a la segurización del canal y de la tokenización. EL trabajo puede ver en Github (s.f.-a).

6.1 Cliente

El cliente responde a las necesidades de la aplicación cumpliendo los requisitos de la gestión contable básica. Para ello nos apoyaremos en las funcionalidades para la gestión de apuntes contables para los diferentes ejercicios en los que el cliente tenga permiso asignado. Además de poder introducir apuntes en el libro borrador (por defecto), podrá pasar todos estos al general y de esta manera poder proceder con la creación de listados.

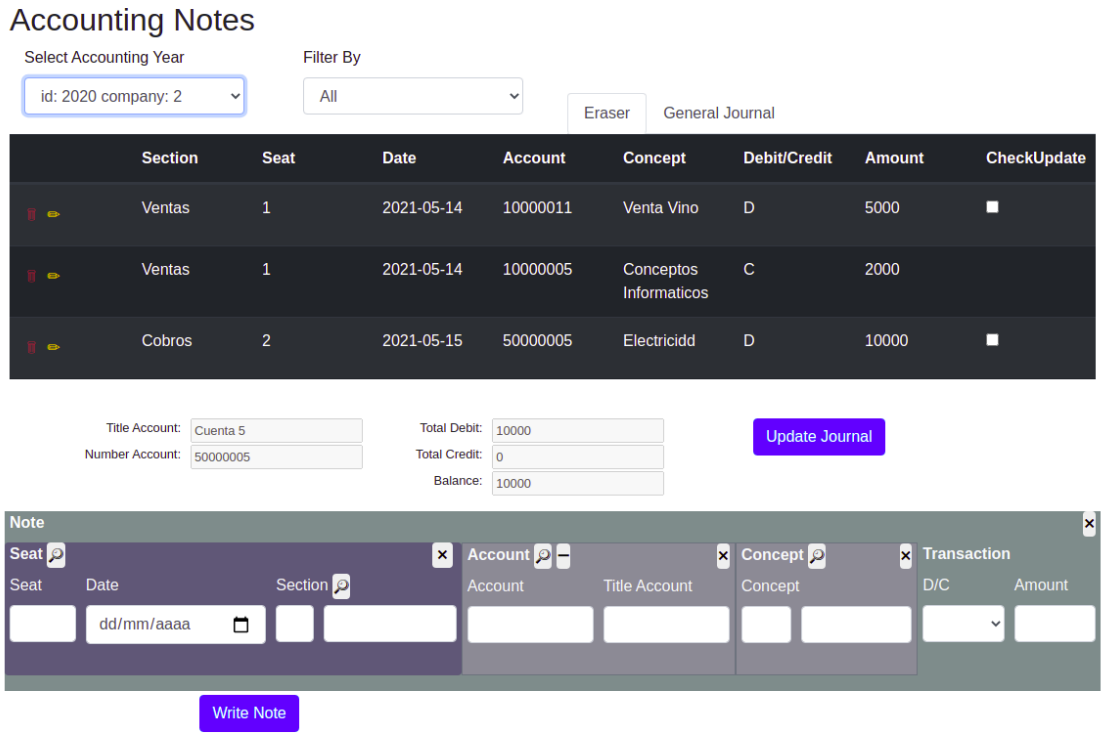

Figura 6.1: Formulario apuntes contables.

Los listados se generarán de 2 maneras, por apuntes contables del libro general y por

cuentas del libro general.

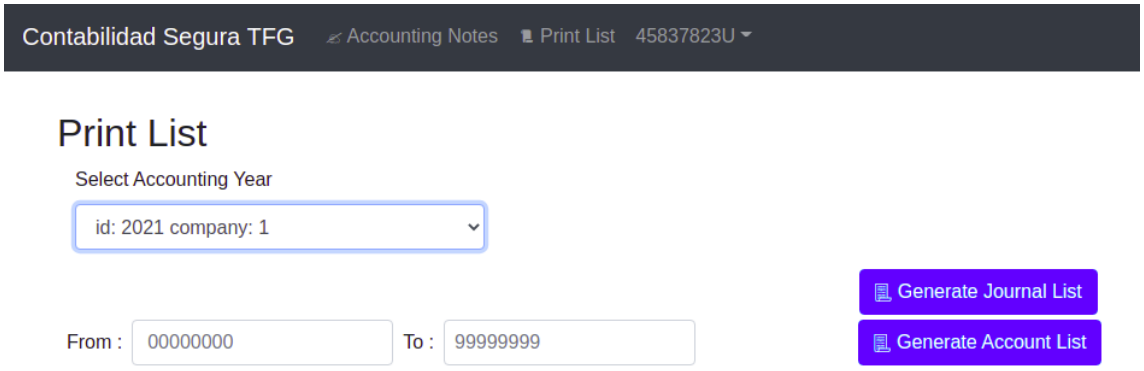

Figura 6.2: Formulario listados.

La gestión del administrador será la encargada de dar de alta las empresas/ejercicios contables (ver **figura 6.3**), usuarios (ver **figura 6.4**) y otorgar los permisos a los usuarios para poder gestionar empresas (ver **figura 6.5**).

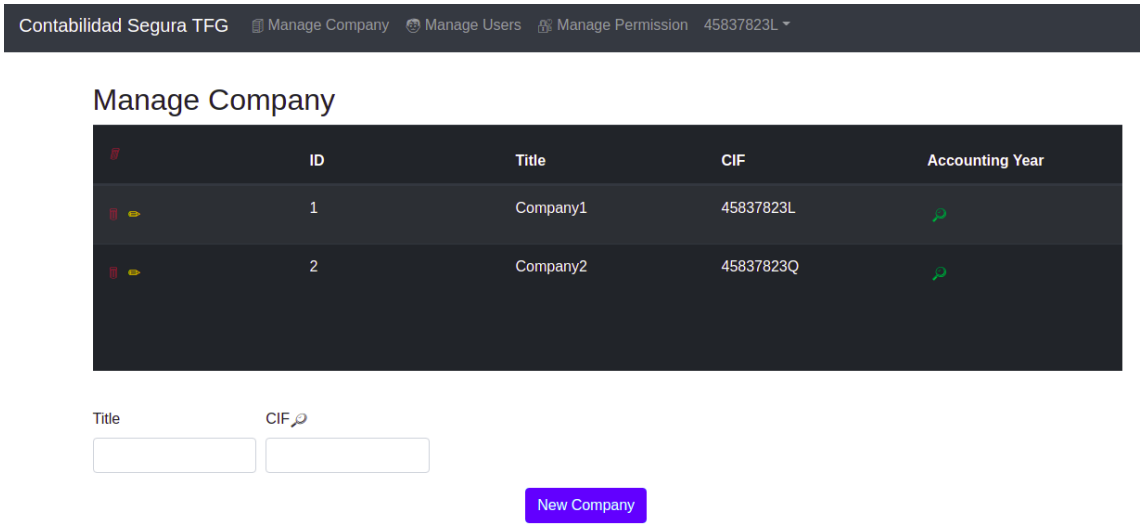

Figura 6.3: Formulario administrador. Gestión empresas.

6.2 Servidor

El servidor funciona como API/REST segura permitiéndonos la segurización de los datos sensibles. Principalmente se consigue con una conexión que guarda las relaciones del diagrama usando *sqlite3* y *sqlcipher*.

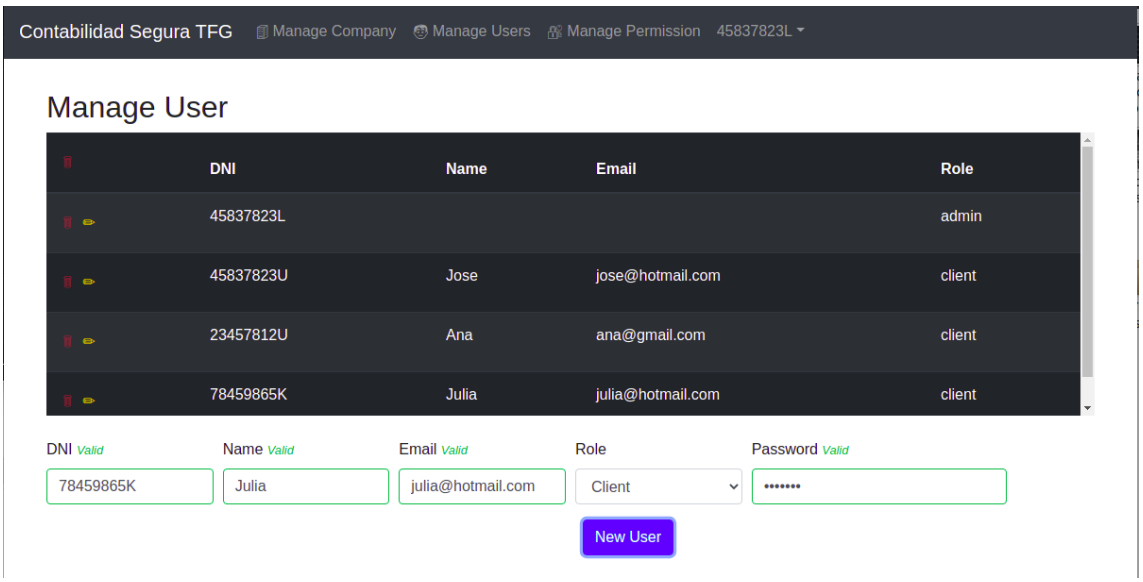

Figura 6.4: Formulario administrador. Gestión usuarios.

El criptograma resultante de usar *sqlcipher* con AES nos proporcionará seguridad ante ataques que puedan acceder al servidor de modo que necesiten de la clave de descifrado para obtener la información en claro.

Además segurizamos el canal para evitar ataques que operen como *man in the middle*[1]. Esta comunicación será segura gracias al protocolo HTTPS.

[1]Ataques que permiten al atacante la lectura, inserción y modificación a los mensajes enviados en una comunicación con la intención de obtener algún beneficio, como por ejemplo hacerse pasar por uno de los actores en al comunicación.

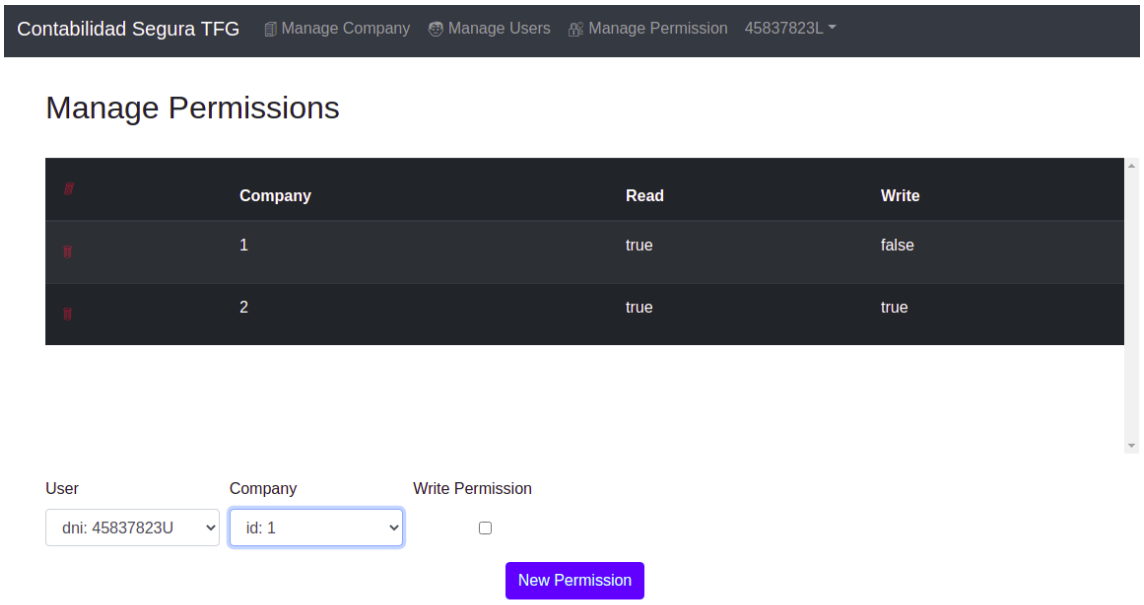

Figura 6.5: Formulario administrador. Gestión permisos.

7 Conclusiones y futuras líneas de mejora

Podemos concluir que se han cumplido los objetivos dados en el proyecto, tanto en la manera de implementarlos como en los resultados obtenidos. Se ha logrado una aplicación contable básica y segura mediante técnicas criptográficas que podrá servir para la gestión contable de pequeñas empresas. Esta podrá servir como punto de comienzo para la realización de una contabilidad mucho más profunda.

En el proceso de desarrollo se han analizado los aspectos principales a tener en cuenta en lo que a una aplicación básica REST se refiere, añadiendo funcionalidades necesarias a día de hoy como es la tokenización y asignación de roles a los usuarios.

Como futuras líneas de trabajo podrían estar aumentar el numero de funciones en la contabilidad y dar soporte a diferentes ámbitos empresariales dentro de la contabilidad como generación de mas tipos de informes, y tener mayor control a la hora de optimizar espacio en la base de datos.

Bibliografía

Cox, T. J., D'antonio, P., y Schroeder, M. (2005). Acoustic absorbers and diffusers, theory, design and application. *The Journal of the Acoustical Society of America*, *117*(3), 988–988.

FacturaScripts. (s.f.). *Página principal de facturascripts.* Descargado de `https://facturascripts.com/`

git. (s.f.). *Acerca del control de versiones.* Descargado de `https://git-scm.com/book/es/v2/Inicio---Sobre-el-Control-de-Versiones-Acerca-del-Control-de-Versiones`

Git. (s.f.). *Repositorio pdf-simple-sign.* Descargado de `https://git.janouch.name/p/pdf-simple-sign`

Github. (s.f.-a). *Repositorio de este proyecto.* Descargado de `https://github.com/car42ua/Contabilidad-Segura-TFG`

Github. (s.f.-b). *Repositorio gofpdf.* Descargado de `https://github.com/jung-kurt/gofpdf`

Github. (s.f.-c). *Repositorio go-sqlcipher.* Descargado de `https://github.com/mutecomm/go-sqlcipher`

Github. (s.f.-d). *Repositorio jwt-go.* Descargado de `https://github.com/dgrijalva/jwt-go`

Github. (s.f.-e). *Repositorio lorca.* Descargado de `https://github.com/zserge/lorca`

HTTPS/TLS. (s.f.). *Certificados en https/tls.* Descargado de `https://weseblog.com/en/2018/12/11/how-ssl-certificates-work/`

López, M. J. (2021). *Criptografía y seguridad en computadores.*

RAE. (s.f.). *Definición cifrar según la real academia española.* Descargado de `https://dle.rae.es/cifrar`

Wikipedia. (s.f.-a). *Algoritmo aes.* Descargado de `https://es.wikipedia.org/wiki/Advanced_Encryption_Standard`

Wikipedia. (s.f.-b). *Definición de criptoanálisis.* Descargado de `https://es.wikipedia.org/wiki/Criptoan%C3%A1lisis`

Wikipedia. (s.f.-c). *Definición de criptografía.* Descargado de `https://es.wikipedia.org/wiki/Cifrado_(criptograf%C3%ADa)`

Wikipedia. (s.f.-d). *Definición de criptología.* Descargado de `https://es.wikipedia.org/wiki/Criptolog%C3%ADa`

Wikipedia. (s.f.-e). *Definición de phishing.* Descargado de `https://es.wikipedia.org/wiki/Phishing`

Wikipedia. (s.f.-f). *Definición de scrum.* Descargado de `https://es.wikipedia.org/wiki/Scrum_(desarrollo_de_software)`