Universidad de Valladolid

DigIdCom:
GESTIÓN DE IDENTIDAD PARA COMUNIDADES DE USUARIOS

ESCUELA DE INGENIERÍA INFORMÁTICA
MENCIÓN EN TECNOLOGÍAS DE LA INFORMACIÓN

TRABAJO DE FIN DE GRADO

Alumno:
Aarón Montoya Rodríguez

Tutores:
Mercedes Martínez González
Amador Aparicio de la Fuente

Dedicado a todos ellos

Antes de que puedan comenzar a leer esta memoria, quiero poder dar las gracias a todas esas personas que durante estos 6 años me han apoyado y motivado para que no me rindiera, han sido años duros y tengo muy claro que sin ellos no estaría hoy donde estoy, en especial quiero dedicárselo a mis padres Ángel y Ana, a mi hermano Jaime, a mi familia, a mis amigos Diego y Anil, a mis queridos compañeros del Cartoncillo, a mis compañeros de facultad que tantas veces me han salvado en mis bloqueos, a mis dos tutores que a pesar de las dificultades han continuado ayudándome y en estos últimos meses a los que se han vuelto unos inseparables compañeros, mi querido BecaTeam y mis compañeros de BSA.

Resumen

La importancia de la gestión de la identidad digital en la sociedad digital en la que en estos días vivimos y que tan caótico puede llegar ser genera una necesidad clara. ¿Cómo podemos ser capaces de cooperar con diversas personas sin el temor a sufrir un robo de identidad y garantizando que las diferentes personas con las que trabajamos verdaderamente son quien dicen ser?.

El objetivo de este proyecto es obtener un sistema con la capacidad de gestionar la identidad digital de los usuarios dentro de una organización en función de las comunidades a las que estos pertenezcan dentro de dicha organización.

Para ello el sistema hará uso de la infraestructura de clave pública (PKI) y los certificados digitales generados por esta misma infraestructura para distintos usuarios.

Abstract

The importance of digital identity management in the digital society in which we live these days and how chaotic it can be generates a clear need. How can we be able to cooperate with diverse people without the fear of identity theft and guaranteeing that the different people we work with are truly who they say they are?

The objective of this project is to obtain a system with the ability to manage the digital identity of users within an organization based on the communities to which they belong within said organization.

For this, the system will make use of the public key infrastructure (PKI) and the digital certificates generated by this same infrastructure for different users.

Capítulo 1 - Introducción

Este capítulo quiere servir como preámbulo para poder llegar a comprender el valor del sistema que hemos tenido oportunidad de construir. Introduciremos la motivación y objetivos de nuestro trabajo, así como la organización de esta memoria.

1. Motivación

A día de hoy nos encontramos frente a una sociedad prácticamente digitalizada y la gestión de identidad digital dentro de ella se ha vuelto un verdadero problema. Muchas de las gestiones que realizamos en nuestro día a día exigen conocer quién es aquel que realiza esa gestión, esto naturalmente no pasa desapercibido para aquellos que desean aprovecharse de otros, por lo que gestionar de la manera más correcta y fiable la identidad de las personas físicas dentro de un mundo digital ya no es un capricho sino una necesidad.

Esto que comentamos se ve a mayores incrementado si hablamos del ámbito empresarial o de investigación, donde es muy habitual encontrarnos con equipos que involucran diversas comunidades de usuarios. Cada usuario dentro del equipo puede a su vez pertenecer a más de una comunidad, lo cual nos deriva, una vez más, en la necesidad de gestionar de manera adecuada identidades asociadas a comunidades.

2. Objetivos

El principal objetivo de este proyecto es lograr la construcción de un sistema con capacidad para gestionar la identidad digital de un grupo de usuarios pertenecientes a una organización, teniendo en cuenta como variables las distintas comunidades a las que estos puedan pertenecer. Para ser capaces de lograr esto emplearemos infraestructura de clave pública (PKI).

Otro de los objetivos para nuestro sistema es ser capaz de soportar las variaciones en la situación actual de la organización a la cual esté dando servicio de manera que pueda continuar con su labor de gestión sin complicaciones.

3. Organización de la memoria

Este documento se organiza en siete capítulos de la siguiente manera:

- **Capítulo 1 - Introducción:** Una breve motivación para nuestro proyecto.

- **Capítulo 2 - Planificación del proyecto:** Descripción de la planificación inicial del proyecto, incluyendo aspectos como la planificación de tareas, el plan de riesgos y una pequeña estimación del coste del proyecto.

- **Capítulo 3 - Fundamentación teórica:** Explicación de los conceptos teóricos necesarios para el correcto seguimiento de esta memoria.

- **Capítulo 4 - Análisis:** Descripción del proceso de análisis del proyecto, se especifican los requisitos y los casos de uso que el proyecto deberá cumplir.

- **Capítulo 5 - Diseño:** Se detalla el diseño del sistema construido.

- **Capítulo 6 - Implementación y validación:** Especificación de las tecnologías empleadas en la construcción del sistema, los métodos usados y las pruebas realizadas para comprobar que los resultados obtenidos son los esperados.

- **Capítulo 7 - Conclusiones y líneas futuras:** Reflexión sobre el proyecto realizado y cuales son los siguientes pasos que podrían seguirse con nuestro proyecto.

Capítulo 2 - Planificación del proyecto

1. Plan Inicial

Para el desarrollo de nuestro proyecto principalmente seguiremos una metodología de cascada, esto implica que se desarrollará de manera secuencial, salvo en ciertas etapas donde podamos realizar un desarrollo en paralelo junto con otra fase ya que ambas tienen aspectos en común y de esta forma facilitar el poder llegar más profundo en ciertos detalles.

El motivo de seleccionar esta metodología para el desarrollo de nuestro proyecto, es sacar partido a la sencillez puesto que nuestro equipo de trabajo es muy reducido y de esta manera es mucho más simple realizar la distribución de trabajo. Debemos ser conscientes también de las desventajas del modelo en cascada y es que realizar un cambio en una de las primeras fases, cuando el desarrollo se encuentra en un estado avanzado, puede ser excesivamente costoso por lo que deberemos ser muy cuidadosos durante todo el desarrollo.

Sobre el modelo en cascada existen diversas variantes, como aquellas que emplean el uso de prototipos, fases solapadas, subproyectos o reducción de riesgos.

Podemos clasificar las distintas tareas de nuestro proyecto en 6 fases distintas:

1. **Inicio:** En esta fase asimilamos los conceptos teóricos para la creación de nuestro sistema y planificamos cómo llevar a cabo nuestro proyecto.

2. **Análisis:** Esta fase será la que logre identificar cuáles son los requisitos de nuestro sistema y cuáles son los posibles casos de uso del mismo.

3. **Diseño:** En nuestro proyecto está será la fase más laboriosa y larga, y en ella detallaremos todos los elementos de nuestro sistema y cómo afectan estos a él.

4. **Implementación:** En esta fase pondremos en marcha el sistema diseñado en la anterior fase, se tratara de una implementación a menor escala pero suficiente para comprobar que es escalable a una organización del mundo real, es muy posible que en esta fase se solapen pruebas con la propia implementación de manera que se puedan retroalimentar entre ambas.

.

5. **Fin:** Durante esta fase “puliremos” todos los detalles del proyecto y de la memoria del mismo.

Las fechas inicio y finalización las detallamos a continuación, seguidamente de las tareas junto con su duración estimada y las dependencias que tienen con sus predecesoras (Tabla 1). La planificación puede verse alterada en función de distintos imprevistos que aparezcan es por eso que existe una fecha "máxima" de finalización que permita maniobrar en caso de algún retraso.

- **Fecha de Inicio**: 14 de Febrero de 2022
- **Fecha de Finalización Ideal**: 21 de Junio de 2022
- **Fecha de máxima de Finalización**: 15 de Julio de 2022
- **Duración estimada del proyecto**: 696 horas aproximadamente

El diagrama de Gantt correspondiente a la planificación inicial del proyecto se muestra en la Figura 1. Como se puede ver en este diagrama, todas las tareas tienen dependencias de fin a inicio con sus predecesoras, es decir, cada tarea debe esperar el fin de su predecesora antes de poder iniciarse.

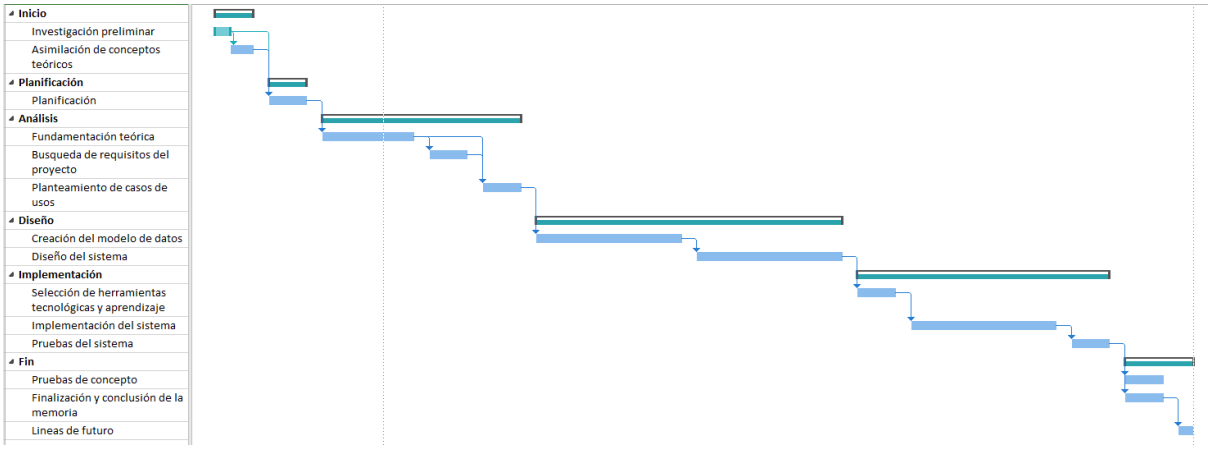


Figura 1: Diagrama de Gantt inicial del proyecto

Como podemos apreciar en el diagrama (figura 1) la tarea que más tiempo nos va a llevar es el diseño del sistema que consta de una duración de 240 horas. Cabe destacar que la totalidad del proyecto coincide en el tiempo con la realización de las prácticas en empresa del alumno por lo que podría limitar el tiempo disponible.

Nombre de tarea	Duración	Comienzo	Fin	Predecesoras
INICIO				
1:Investigación preliminar	16 horas	lun 14/02/22	mar 15/02/22	
2:Asimilación de conceptos teóricos	24 horas	mié 16/02/22	vie 18/02/22	2
PLANIFICACIÓN				
3: Planificación	40 horas	lun 21/02/22	vie 25/02/22	2;3
ANÁLISIS				
4: Fundamentación teórica	80 horas	lun 28/02/22	vie 11/03/22	3
5: Búsqueda de requisitos del proyecto	40 horas	lun 14/03/22	vie 18/03/22	4
6: Planteamiento de casos de usos	40 horas	lun 21/03/22	vie 25/03/22	4;5
DISEÑO				
7: Creación del modelo de Datos	120 horas	lun 28/02/22	vie 15/04/22	6
8: Diseño del sistema	120 horas	lun 18/04/22	vie 06/05/22	7
IMPLEMENTACIÓN				
9: Selección de herramientas tecnológicas y aprendizaje	40 horas	lun 09/05/22	vie 13/05/22	8
10: Implementación del sistema	40 horas	lun 16/05/22	vie 03/06/22	9
11: Pruebas del sistema	40 horas	lun 06/06/22	vie 10/06/22	10
FIN				
12: Pruebas de concepto	40 horas	lun 13/06/22	vie 17/06/22	11
13: Finalización y conclusión de la memoria	40 horas	lun 13/06/22	vie 17/06/22	11
14: Ampliaciones de futuro	16 horas	lun 20/06/22	mar 21/06/22	13

Tabla 1: Planificación del proyecto

2. Plan de Riesgos

En este apartado analizaremos los distintos riesgos que podríamos enfrentar en el desarrollo de nuestro proyecto, para ello en primer lugar identificaremos los posibles riesgos y en segundo lugar veremos la exposición que tenemos frente a ellos con la ayuda de una matriz de riesgos por impacto y frecuencia, y para terminar veremos los diferentes planes de contingencia de aquellos riesgos clasificados como altos o superiores para mitigar el impacto en caso de que se materializaran.

Para la evaluación de la frecuencia e impacto usaremos una escala de 1 a 5, siendo 1 lo más bajo y 5 lo más alto, y en función del valor de la combinación de estos dos valores obtendremos la exposición a los diferentes riesgos. La matriz de clasificación quedaría de la siguiente manera (Figura 2).

FRECUENCIA	5	Alta	Alta	Muy Alta	Muy Alta	Muy Alta
	4	Media	Media	Alta	Muy Alta	Muy Alta
	3	Baja	Media	Media	Alta	Muy Alta
	2	Baja	Baja	Media	Media	Alta
	1	Baja	Baja	Baja	Media	Alta
		1	2	3	4	5
		IMPACTO				

Tabla 2: Matriz de clasificación de riesgos

Ahora identificaremos cada riesgo, lo clasificaremos tal y como hemos explicado anteriormente y aportaremos su correspondiente plan de contingencia en caso de ser necesario y dentro de nuestros límites.

- **Fallo de planificación:**

Impacto: 4 *Frecuencia: 4* *Exposición: Muy Alta*

Para reducir la probabilidad de que este riesgo se materialice introducimos un margen de tiempo en la planificación de nuestro proyecto.

- **Cambios de horario de otras actividades:**

Impacto: 5 Frecuencia: 1 Exposición: Alta

En caso de que este riesgo se materialice para solventarlo deberemos de replanificar las tareas referentes al proyecto.

- **Daño en el ordenador de trabajo:**

Impacto: 5 Frecuencia: 1 Exposición: Alta

Para evitar mitigar lo más posible este riesgo intentaremos trabajar al máximo posible con herramientas cloud como google docs, en caso de no ser posible documentaremos el trabajo realizado y guardaremos copias de seguridad en almacenamiento cloud como google drive.

- **Enfermedad:**

Impacto: 4 Frecuencia: 3 Exposición: Alta

Debemos prestar atención a esto y más en los tiempos que corren con la pandemia en la que vivimos, de ahí que su frecuencia sea algo más elevada de lo normal. Para contrarrestar este riesgo la solución será introducir un margen de tiempo en nuestra planificación.

- **Cambios en los requisitos:**

Impacto: 3 Frecuencia: 2 Exposición: Media

Aumentaremos el tiempo dedicado al análisis de requisitos para poder evitar que puedan aparecer variaciones de estos en el futuro, también reducirá considerablemente la aparición de este riesgo reporte de avances a los tutores del proyecto.

- **Fallo tardío de diseño:**

Impacto: 5 Frecuencia: 2 Exposición: Alta

La única manera en la que podremos mitigar el daño que suframos con la materialización de este riesgo será contar con el suficiente margen de tiempo en la planificación como para corregir el fallo encontrado, las reuniones con los tutores también ayudarán en la reducción de probabilidades de que este riesgo aparezca.

- **Falta de conceptos teóricos o prácticos:**

Impacto: 4 *Frecuencia: 2* *Exposición: Media*

Aumentaremos el tiempo invertido en el estudio de estos elementos para asimilar de manera correcta todo lo necesario para la elaboración del proyecto.

3. Presupuesto

Como en todo proyecto existen unos gastos necesarios para su desarrollo este apartado tiene como finalidad intentar realizar un pequeño presupuesto, teniendo en cuenta los gastos del personal implicado y los productos y herramientas empleados.

El gastos en productos es nulo ya que todo el software que empleamos es de código libre desde el sistema operativo hasta las herramientas. Sin embargo el hardware que empleamos si tiene un coste de alrededor de unos 500€ el precio medio de una computadora normal.

El gasto de personal en este caso es 0€ ya que el desarrollador del sistema no cobrará por la realización de este proyecto. Sin embargo, podemos estimar el costo que supondría la contratación de un desarrollador junior.

Un desarrollador junior en España cobra unos 18.000 € al año, con una jornada de 40 horas semanales, es decir unas 160 horas al mes, lo que en un año supondría 1920 horas, de lo que podemos obtener que el sueldo sería de unas 10 € por cada hora de trabajo. Lo que nos daría un total de 6.960 € aproximadamente.

Capítulo 3 - Fundamentación teórica

1. Gestión de identidad

Entendemos la gestión de identidades (IDM, Identity Management) como el primer paso dentro de un proceso mayor conocido como gestión de identidades y accesos (IAM, Identity and Access Management). El IDM únicamente se encarga de "¿Cómo identificamos al usuario?", mientras que el proceso de IAM se preocupa también de a que recursos y/o sistemas accede el usuario. **[6]**

El proceso de IDM busca autentificar de la manera más fuerte posible, en la medida de los recursos que la organización disponga, que un usuario verdaderamente es quien dice ser. Para lograr esto existen diferentes técnicas que podremos clasificar en tres grupos diferentes en función de cómo se relacionan con el usuario.

- **Algo que el usuario SABE .** Esta sería una de las técnicas más tradicionales, la finalidad es identificar a un usuario mediante algo que únicamente conoce este, dentro de este grupo podríamos colocar las tradicionales contraseñas o preguntas de seguridad.

- **Algo que el usuario TIENE.** Estas técnicas pretenden identificar mediante la posesión de algún tipo entidad, puede tratarse de un objeto físico, un fichero digital o una combinación de ambos, en este grupo podemos encontrar tanto tecnologías de autenticación mediante multifactor(las cuales consistente en enviar un código variable al usuario a un dispositivo que es de su propiedad), tarjetas de acceso y certificados digitales, que en nuestro proyecto será en lo que nos centraremos.

- **Algo que el usuario ES.** En nuestro tercer grupo podemos encontrar todas aquellas tecnologías que permiten la identificación del usuario mediante aquello que físicamente como persona es, en este grupo entrarían todas aquellas tecnologías biométricas como son escáneres de retina, sensores de huella dactilar o escáneres faciales (como los empleados en gran mayoría de dispositivos móviles actualmente).

Habiendo explicado estos tres grupos podemos decir que verdaderamente ninguno de ellos por sí solo es capaz de lograr una autenticación fuerte del usuario, pero si realizamos la combinación de al menos dos de estos grupos podemos asegurar que el usuario es quien dice ser, naturalmente los recursos de los que disponga la organización serán el factor limitante de estos sistemas, pongamos un ejemplo, en una organización no presenta el mismo coste realizar una identificación combinada mediante una contraseña y código multifactor, que realizarla mediante la combinación de una contraseña y un sensor de huella dactilar que deberá distribuir a todos los miembros de la organización, obviamente la implementación de un sistema que a grandes rasgos únicamente necesita un componente software (puesto que es

bastante habitual recibir los códigos multifactor en el móvil personal del usuario) y por tanto es menos costoso de mantener y variar su volumen, que aquel que necesita la distribución de un hardware especial para todos los miembros de la organización sumado al componente software para soportar a mayores todo este sistema.

Pero bien ahora nos falta saber por que el IDM es tan necesario e importante para una organización, para comprender esto únicamente deberemos hacernos una pregunta "¿Qué pasaría si alguien logra hacer ver al sistema que es otra persona?" , para responder esta pregunta vamos a plantear la siguiente situación: Mallet es un trabajador de una empresa encargada de la distribución de material médico para su zona geográfica, en esta empresa únicamente poseen un sistema de identificación mediante contraseña, tras varios intentos de promoción sin éxito para un puesto mejor dentro de la empresa, Mallet decide que es momento de marcharse de la empresa, pero antes quiere que la empresa pague por no "valorar su talento" por ello logra hacerse con la contraseña de su compañero Bob, que trabaja en el departamento de distribución y es un poco despistado por lo que siempre deja su contraseña anotada en un papel pegado en su portátil, Alice accede al sistema con la contraseña de Bob extrae y elimina del sistema los datos de distribución de los últimos 5 años; tras esto Mallet difunde con las credenciales de Bob la información, cierra sesión y se marcha. Las consecuencias de esto son incontables: pérdida de información, difusión de datos privados, bloqueo temporal de la distribución del material, suministros insuficientes para centros sanitarios, pérdida de reputación para la empresa, pérdida económica para la empresa, … y sin olvidar al pobre de Bob que en un primer momento será el único responsable puesto que a ojos del sistema fue el que realizó todas estas acciones. Este ejemplo ilustra de manera adecuada el "valor" que tiene una identidad dentro de una organización y por qué deberíamos velar por gestionarlas de manera apropiada. **[2]**

2. Estructura PKI para gestión de identidad

La *Public Key Infrastructure* (PKI) o Infraestructura de Clave Pública, en español, consiste en conjunto de servicios y componentes informáticos que permiten la gestión, control y administración de toda clase de certificados digitales.

En resumidas cuentas la Infraestructura de Clave Pública es un complejo combinado de software y hardware aplicado en políticas y tareas de seguridad digital. Pero el punto que hace destacar PKI es la posibilidad que ofrece de integrar los certificados digitales junto a la criptografía de Clave Pública.

Teniendo conocimiento de los dos puntos claves de PKI (certificados digitales y criptografía de Clave Pública) ahora pasaremos a explicar cada uno de ellos. **[5] [9] [16]**

2.1 Criptografía de Clave Pública

Para que podamos entender en qué consiste un sistema de criptografía de clave pública en primer lugar deberemos saber que es un sistema criptográfico sin ningún tipo de adjetivo añadido. Según la RAE definimos criptografía como:

"*Arte de escribir con clave secreta o de un modo enigmático"* **[1]**

Por lo tanto un sistema criptográfico es aquel que permite cifrar y descifrar información empleando un algoritmo, una o unas claves, texto plano y el correspondiente texto cifrado.

El sistema de criptografía de clave pública es aquel en el que las claves, o llaves, vienen por pares una se mantiene privada, y solo es conocida por su propietario, mientras que la otra se mantiene pública y está disponible para cualquiera. **[4]**

Cuando se emplean para firmas digitales, la clave privada es la que se emplea para realizar la firma y la clave pública la encargada de verificar. Esto significa que cualquier usuario de la organización puede verificar una firma, puesto que exclusivamente el dueño de la clave privada podría generarla.

Cuando se usa para realizar un cifrado, la clave pública es la encargada de realizar el cifrado mientras que la clave privada será la encargada de descifrar. Esta es una de las principales ventajas frente a un sistema criptográfico de clave simétrica, en el que la clave de cifrado es la misma en ambos extremos lo cual complica la tarea de que ambos extremos obtengan de manera segura esta clave, ya que al tratarse de una clave con caracter público, no es necesario ocultarla y cualquier persona podría cifrar un mensaje, pero únicamente el propietario de dicha clave podría descifrarlo. Naturalmente no todo es tan sencillo y positivo, ya que esto hubiera provocado la desaparición del cifrado simétrico y esto es bien sabido por

todos que no es así, los algoritmos asimétricos son mucho más lentos y no permiten cifrar mensajes tan largos puesto que el tamaño máximo capaz de cifrar es proporcional al tamaño de la clave. Como resultado de este tipo de desventajas es muy habitual combinar ambos sistemas para compensar las deficiencias de cada uno, el cifrado simétrico se encarga del mensaje mientras que el cifrado asimétrico se encarga de transmitir la clave simétrica que se ocupará del mensaje. El siguiente esquema muestra un ejemplo de cómo funciona este sistema combinado.

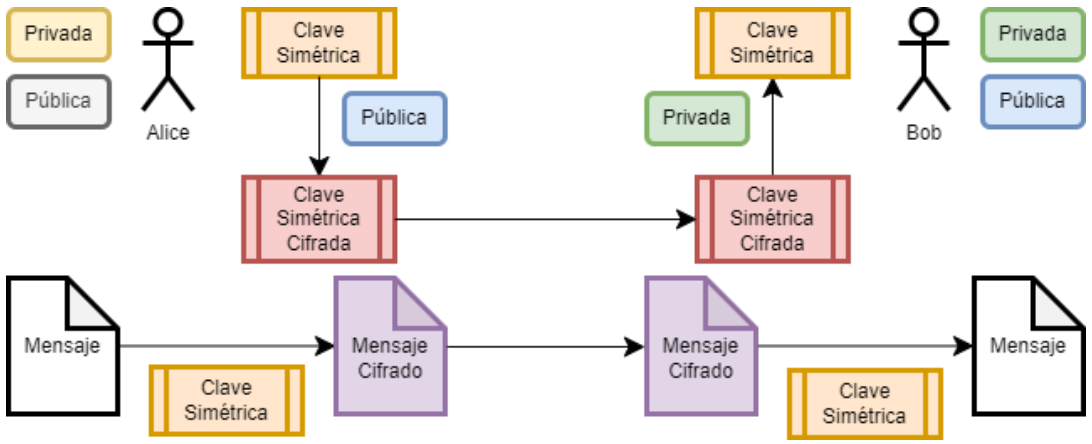


Figura 2: Combinación de los sistemas criptográficos simetrico y asimetrico para la transferencia de un mensaje

2.2 Certificados digitales

Como hemos comentado anteriormente los certificados digitales son el segundo gran punto de PKI y ahora veremos cuales son las características de estos y los distintos usos que estos tienen. Pero antes de eso debemos definir los términos de **Firma Digital** y **Autoridad Certificadora** (CA), Ambos conceptos son claves a la hora de comprender los certificados digitales.

Una **Firma Digital** se trata de un sello de autenticación electrónico que busca lograr la cualidad de no repudio y de integridad, esto quiere decir que ese sello únicamente ha podido ser colocado por una entidad concreta, veamos como funciona con la siguiente figura.

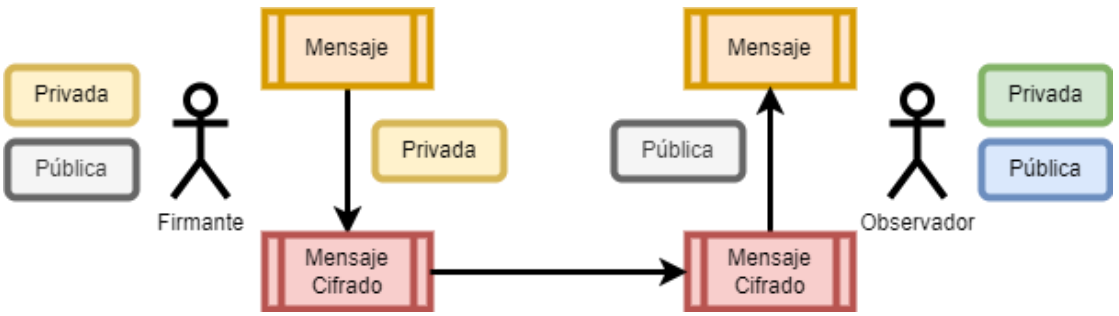


Figura 3: Funcionamiento de la firma digital

Para conseguir la cualidad de no repudio, y de esta forma garantizar que el firmante es quien afirma ser, lo que nuestro firmante hara sera cifrar el mensaje en cuestión con su clave privada para que de esta forma el observador los descifre con la clave pública del firmante y de esta forma confirmar la identidad del firmante.

Una **Autoridad de Certificación** (CA) es una entidad de confianza que emite y revoca certificados digitales utilizados en transacciones y firmas electrónicas. Esta confianza se logra gracias a la imagen de la CA, que actúa como parte interviniente en la relación entre los diversos miembros de una organización. Por lo tanto, cuando se realice cualquier transacción entre dos partes, la CA confiará en los documentos que se gestionen y firmen por no ser una parte relacionada.

Algunas de las funciones de una CA son proporcionar servicios tales como emitir certificados, revocar listas de certificados [1], verificar la validez de los certificados, etc. Además, la CA registra la fecha y la hora exactas en que se firmó electrónicamente el documento, lo que se denomina sello de tiempo. **[15]**

Teniendo claros estos dos conceptos que son parte fundamental de los certificados digitales estamos más que preparados para comprender en qué consisten, qué características tienen y cual es el tipo más interesante para nuestro proyecto.

Un certificado digital es un archivo informático firmado electrónicamente por un prestador de servicios de certificación y considerado por otras entidades como la autoridad de dicho contenido, vincula los datos de verificación de la firma al firmante, de manera que solo ese firmante puede firmar, y confirmar su identidad. Tiene una estructura de datos que contiene información sobre una entidad (por ejemplo, una clave pública, una identidad o un conjunto de permisos). **[18]**

En función del uso que le queramos dar a nuestros certificados contendrán una información u otra, y esta será la que da lugar a los distintos tipos de certificados que existen.

- **Certificados de Clave Pública:** Esta clase será la que emplearemos para el desarrollo de nuestro proyecto, ya que principalmente estos son empleados en protocolos autenticación de entidades. Estos certificados asocian una clave pública a una identidad, la clave privada pareja de dicha clave pública se encontrará en posesión de la entidad con la identidad especificada en el certificado. Además, se podrá incluir otra información relevante, como el periodo de validez, el identificador del algoritmo que puede utilizar la clave pública, o las políticas o restricciones de uso del certificado.

[1] *Cuando un certificado es revocado implica que este pierde su validez y por lo tanto no garantiza la identidad del propietario, este puede revocarse por dos motivos, se ha visto comprometido o su tiempo de vida ha terminado*

- **Certificados de Atributos:** Son empleados normalmente en protocolos relacionados con la autorización de entidades. En líneas generales no contienen claves públicas, en lugar de eso relacionan otra clase de información como roles o privilegios.

2.2.1 X.509: Certificados Digitales y codificaciones DER, CRT y CER

Un certificado digital X.509 es aquel que ha sido firmado digitalmente y/o codificado siguiendo la RFC 5280 **[13] [8]**.

La única finalidad del estándar X.509 es definir la sintaxis del certificado, esto implica que no se encuentra no tiene necesidad de emplear ningún algoritmo en específico, y los campos que podemos encontrar dentro de su estructura son los siguientes:

- **Versión:** X.509 dispone de tres versiones diferentes siendo la v3 la más reciente de todas las existentes hasta la fecha.

- **Serial Number:** (Número de serie) Se trata del identificador único para todos los certificados emitidos por una misma autoridad certificadora (CA).

- **Algoritmo empleado en la firma digital:** Identifica qué algoritmo empleo la CA del certificado para firma el mismo.

- **Nombre del certificador:** Nombre de la CA.

- **Tiempo de vida del certificado:** Periodo de validez del certificado, pasado este tiempo el certificado quedará invalidado.

- **Propietario:** Nombre del dueño del certificado en cuestión.

- **Clave pública del propietario:** Clave pública del dueño, y algoritmo asociado.

- **Extensiones:** Información adicional permitida por X.509.

- **Firma digital de la CA:** Firma digital generada por la certificadora que respalda todo lo anterior.

Lo que vemos en la figura 3 se trata de un certificado X.509, que sirve como ejemplo para ilustrar los campos anteriores y no se trata de uno de los que existirá en nuestro sistema.

```
Certificate:
Data:
Version: 3 (0x2)
Serial Number: 9 (0x9)
Signature Algorithm: sha1WithRSAEncryption
Issuer: C=US, ST=Florida, L=Tampa, O=Text CA,
CN=bsd.jcs.local
Validity
Not Before: Mar 31 20:02:42 2003 GMT
Not After : Mar 30 20:02:42 2004 GMT
Subject: C=US, ST=Florida, L=Tampa, O=sslecho,
CN=linux.jcs.local
Subject Public Key Info:
Public Key Algorithm: rsaEncryption
RSA Public Key: (1024 bit)
Modulus (1024 bit):
00:d6:6f:d6:40:00:8a:c6:86:00:b8:31:62:f3:a6:
bd:c1:f0:10:b5:69:34:8c:f7:d6:09:98:66:9d:dd:
a5:90:5e:58:fb:06:9d:59:21:75:fb:ac:ab:86:56:
83:57:af:85:1e:53:90:45:f7:e9:3f:66:b1:f3:e7:
fd:59:c1:88:ee:86:13:3c:79:55:c9:50:58:ae:5a:
32:d5:6e:aa:a7:f0:fd:2c:88:b9:89:1c:9d:3e:95:
27:6b:cc:a9:1f:5e:c0:99:d5:65:79:1e:2d:64:d3:
63:dd:99:8f:1f:22:1d:2f:2e:1b:f9:39:6c:c5:1e:
b3:01:a0:1a:07:56:21:5b:c3
Exponent: 65537 (0x10001)
X509v3 extensions:
Netscape Cert Type:
SSL Server
Signature Algorithm: sha1WithRSAEncryption
4b:e7:22:94:f9:a9:c3:db:6b:a5:c3:ea:39:b7:9a:04:36:c9:
de:d7:c2:ed:59:d7:bb:b9:4c:ec:35:a4:15:e9:32:d6:b0:ea:
d8:64:5d:5c:41:3f:bb:c9:41:c7:32:fd:ad:47:52:20:c4:d5:
04:3a:92:a8:59:f8:34:3c:57:bd:cc:15:ac:f4:3e:59:11:3f:
c4:3f:2f:a5:7f:ef:89:8f:13:51:e6:9c:a7:94:20:71:ed:5a:
1d:57:65:bb:38:34:2f:0a:86:73:e2:18:e0:8f:23:4d:d0:a3:
37:b6:ee:0f:44:07:1d:94:66:70:78:ef:31:d1:97:50:11:ec:
25:c3
```

Figura 4: Certificado digital X.509

Ahora bien X.509 dispone de diferentes extensiones y codificaciones (DER, PEM, CRT, CER, …). Cada una de estas tiene sus propias características y no pueden ser intercambiadas sin esperar que algún problema surja.

Las codificaciones que vamos explicar a continuación en ocasiones también pueden ser empleadas como extensiones en los archivos de certificados digitales:

- **PEM:** Por norma general son empleados en certificados digitales X.509 v3 y están codificados en Base64 / ASCII. La manera más sencilla de identificarlos es por que al abrirlos con un editor de texto con líneas similares a:

```
-----BEGIN CERTIFICATE-----
...
-----END CERTIFICATE-----
```

- **DER:** Esta extensión es empleada para certificados digitales codificados en forma binaria. Suelen tener extensión de tipo CRT o CER. Tenemos que destacar que DER se trata de una codificación de certificados X.509 y no de un tipo de certificado.

Las extensiones que analizaremos a continuación trabajan junto con las codificaciones que hemos explicado anteriormente y son las más empleadas a día de hoy.

- **CRT:** Esta extensión permite tanto codificaciones binarias DER o codificaciones Base64 / ASCII PEM. Esta extensión es la más empleada en sistemas Unix o Linux.

- **CER:** Se trata de una alternativa a CRT pero para Microsoft. Internet Explorer también reconoce la extensión CER como un comando para ejecutar ejecutables de la API criptográfica de Microsoft.

- **KEY:** Para los pares de claves esta extensión es la utilizada, pueden estar codificadas en binario DER o Base64 / ASCII PEM.

A la vista de esto el único caso que permite intercambiar las extensiones CRT y CER sin ningún tipo de problema es cuando la codificación es compatible entre las extensiones, por ejemplo podríamos cambiar entre CER y CRT siempre y cuando la codificación fuera PEM.

Capítulo 4 - Análisis

Durante esta fase hemos especificado de manera detallada todas aquellas funcionalidades que existen dentro de nuestro sistema.

Pero para ello previamente vamos a identificar con que usuarios y grupos vamos a trabajar en nuestro sistema.

1. Tipos de usuarios

A la hora de analizar las funcionalidades hemos podido identificar dos clases distintas de usuarios que podrán estar dentro del sistema que nos encontramos desarrollando. Los distintos tipos de usuarios que encontraremos son:

- **Miembro:** Persona perteneciente a la organización sobre la cual está implantado nuestro sistema. Lo único que realizará esta clase de usuarios será recibir los certificados emitidos para que estos se puedan identificar como miembros de las comunidades a las que pertenecen.

- **Gestor:** Miembro de la organización con capacidad de gestionar el sistema implantado en la organización a la cual pertenece. Entendiendo como gestionar la realización de todas las tareas como serían dar de alta y baja miembros de la organización, dar de alta y baja miembros de una comunidad, generar y revocar certificados, crear y mantener las autoridades certificadoras, …

2. Tipos de grupos

Al igual que ha ocurrido con los usuarios también podemos identificar distintas clases de grupos en nuestro sistema, más concretamente lo que podemos encontrar es un grupo global y dentro de este distintos subgrupos la denominación que usaremos será:

- **Organización:** Grupo de personas y medios organizados con un fin determinado sobre el cual podremos implementar nuestro sistema. Nuestro sistema estará implantado en una organización y exclusivamente dará soporte a una, esto es importante destacar para no dar lugar a confusiones.

- **Comunidad:** Subgrupo de personas y medios contenido dentro de nuestra organización, una organización puede tener tantas comunidades como sean necesarias.

3. Requisitos

Para continuar especificaremos aquellos requisitos que nuestro sistema deberá cumplir de manera obligatoria para considerar el uso de este mismo viable, entendemos que un requisito es:

"Capacidad o condición que un usuario considera necesaria para lograr o solventar un determinado objetivo" **[10]**

Clasificaremos los requisitos en tres grupos distintos: funcionales, no funcionales y de seguridad. Debemos tener en cuenta la descripción de términos que hemos realizado anteriormente para evitar confusiones.

3.1 Requisitos funcionales

Consideramos requisitos funcionales como aquellas declaraciones sobre las tareas que nuestro sistema debe ser capaz de realizar.

- **RF1** El sistema debe permitir registrar nuevos miembros.
- **RF2** El sistema debe permitir dar de baja miembros.
- **RF3** El sistema deberá generar certificados digitales que identifiquen a los miembros de la organización.
- **RF4** El sistema deberá de emitir múltiples certificados digitales para un miembro en función de las comunidades de a las que este miembro pertenezca.
- **RF5** El sistema deberá soportar que el usuario gestor revoque los certificados digitales cuyo tiempo de vida se haya superado.
- **RF6** El sistema deberá soportar que el gestor revoque los certificados digitales emitidos que hayan podido verse comprometidos.
- **RF7** El sistema deberá soportar la creación de una nueva comunidad por parte del gestor en caso de que la organización aumente.
- **RF8** El sistema deberá soportar la eliminación de una comunidad por parte del gestor en caso de que la organización se reduzca.
- **RF9** El sistema debe permitir la generación de las claves privadas de los miembros de la organización.

- **RF10** El sistema deberá permitir dar de alta miembros en sus respectivas comunidades.

- **RF11** El sistema deberá permitir dar de baja miembros de sus respectivas comunidades.

- **RF12** El sistema permitirá la existencia de diferentes CA (autoridades certificadoras) para la emisión de los certificados.

- **RF13** El sistema permitirá la existencia de una CA encargada de crear otras CA de menor nivel.

3.2 Requisitos no funcionales

Consideramos requisitos no funcionales como aquellas declaraciones que aplican restricciones a los diferentes servicios o funciones del sistema; en otras palabras describen como el sistema realizará las diferentes tareas.

A fin de lograr que nuestro sistema pueda adaptarse de manera correcta en el futuro no especificaremos tecnologías concretas, sino los requisitos que estas deban cumplir.

- **RNF1** El sistema deberá funcionar en los sistemas operativos más comunes.
    - Hablamos de *Linux* y *Windows* actualmente.

- **RNF2** El sistema deberá emplear tecnología de clave pública.
    - Actualmente *OpenSSL* sería una buena opción.

- **RNF3** El sistema deberá usar extensiones que generen el menor conflicto posible con el hardware encargado de soportar nuestro sistema.

- **RNF4** El sistema deberá usar codificación compatible con la extensión empleada.
    - El tipo de codificación *PEM* sería adecuado en caso de usar extensión *CRT*.

- **RNF5** El sistema solo dará soporte a una organización.

- **RNF6** En el sistema existirá una correlación entre una autoridad certificadora intermedia y una comunidad, una autoridad certificadora solo dará servicio a una comunidad y una comunidad solo dispondrá de una autoridad certificadora.

- **RNF7** En el sistema existirán dos partes una que permita describir el estado actual de la organización donde se aplica y otra que nos permita realizar cualquier tipo de auditoría sobre los distintos miembros que han pasado por la organización o las distintas comunidades que hayan existido.

3.2.1 Requisitos de Seguridad

Podemos considerar estos como requisitos no funcionales, pero resulta interesante darles su propio lugar ya que entendemos los requisitos de seguridad como aquellos que garantizaran la integridad de nuestro sistema, tanto de los datos con los que trata como el sistema en sí mismo.

Al igual que en los requisitos no funcionales no precisaremos tecnologías concretas sino que especificaremos las características que deban cumplir para ser válidas para nuestro sistema.

- **RS1** El sistema únicamente deberá ser administrado por los gestores del sistema y no por cualquier usuario del mismo.

- **RS2** El sistema deberá emplear un algoritmo de cifrado de clave pública seguro.
    - En la actualidad una opción muy válida será *RSA*[2] **[12]**.

- **RS3** El sistema deberá proporcionar claves con longitud suficiente para no ser expuestas.
    - Actualmente 2048 bits serían suficientes.

- **RS4** El sistema deberá usar una función de resumen hash lo menos vulnerable posible.
    - Hoy en día la opción más recomendable sería *SHA256*.

[2] Varios investigadores han estudiado y descubierto cómo obtener la clave privada de un usuario basándose únicamente en la clave pública del mismo. No ataca todo el cifrado RSA-2048 (RSA de 2048 bits), solo las plataformas que lo usan a través de la librería del fabricante Infineon. El cifrado RSA cifra un mensaje en forma digital y para descifrarlo se requieren dos números primos elegidos al azar. Pero para acelerar el proceso, el código base de Infineon construye los números primos subyacentes en claves, por lo que son propensos al proceso de factorización de esos números primos. **[14]**

3.2.2 Requisitos de información

Podemos considerar estos como requisitos no funcionales, pero les daremos su propio lugar para que el futuro modelo de datos que presentaremos sea mucho más claro y sencillo de entender.

- **RI1** Sobre cada miembro recogemos su nombre y apellidos.

- **RI2** Sobre cada gestor recogeremos su nombre y apellidos.

- **RI3** Sobre cada comunidad recabaremos el nombre de la misma, el creador de dicha comunidad, la fecha de creación y el número de miembros.

- **RI4** Sobre cada certificado perteneciente a cada miembro de la organización recogeremos comunidad y organización para las que es válido el certificado, autoridad certificadora que valida el certificado, estado del certificado (revocado o válido), fecha de fin de validez del certificado y número de serie del certificado.

- **RI5** Sobre las diferentes CAs, encargadas de bien emitir los distintos certificados y crear otras CAs, recogeremos datos sobre el nombre de la organización y comunidad para la que emitirá certificados, ubicación geográfica de la comunidad o organización, número de serie de la CA y fecha hasta la cual es válida esa CA.

Para resumir y velar por que no quede posibilidad de duda, únicamente utilizaremos los datos necesarios para crear los certificados de los diferentes elementos de nuestro sistema.

3.2.3 Requisitos de privacidad

Al igual que los anteriores estos también son considerados como requisitos no funcionales. Los requisitos que presentaremos en este apartado son aquellos que aseguren el cumplimiento normativo en materia de privacidad y seguridad de los datos que podamos almacenar.

- **RP1** El sistema solo almacenará los datos estrictamente necesarios.

- **RP2** El sistema eliminará los datos almacenados una vez dejen de ser necesarios, esto implica que cuando un miembro de la organización la abandona sus datos se eliminarán.

- **RP3** El sistema comprobará que los datos han sido eliminados verdaderamente tras estos ser borrados.

- **RP4** El tratamiento realizado de los datos deberá regirse por el RGPD (Reglamento General de Protección de Datos).

4. Casos de uso

Seguidamente analizaremos los diferentes casos de uso que nos encontramos para nuestro sistema. Para ello desglosamos cada caso de uso en una tabla que permita el desarrollo de cada una.

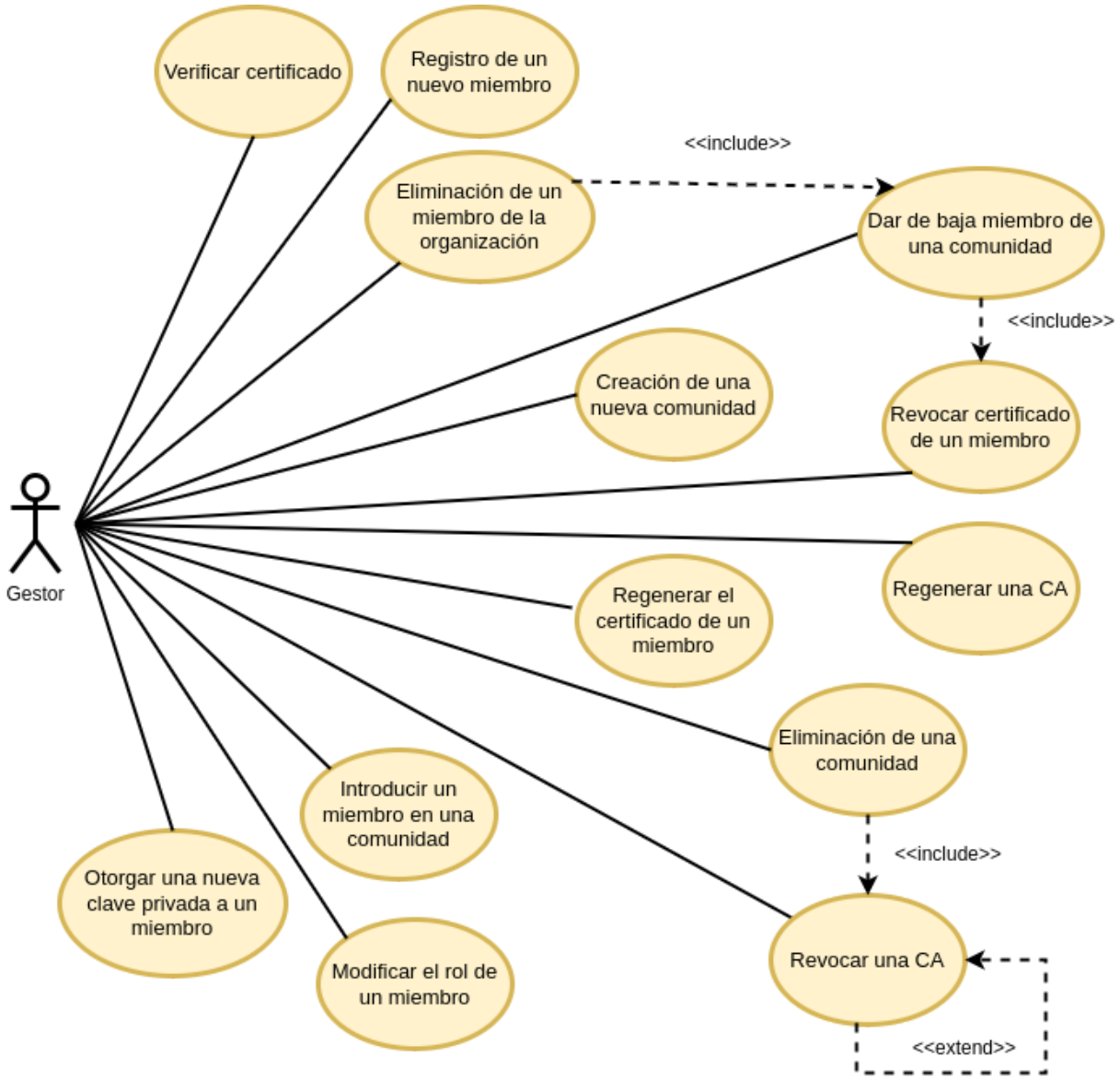


Figura 5: Diagrama de casos de uso del sistema

UC-01	**Registro de un nuevo miembro**
Descripción	El gestor registra un nuevo miembro dentro de la organización.
Precondición	El gestor debe conocer los datos del nuevo miembro.
Secuencia normal	1. El gestor solicita el registro de un nuevo miembro.
	2. El sistema requiere el nombre y apellidos del nuevo miembro y rol del miembro.
	3. El gestor proporciona los datos que han sido solicitados.
	4. El sistema asigna un identificador (ID) al nuevo miembro registrado.
	5. El sistema registra el nuevo miembro.
Flujos alternativos	3a-a. El gestor indica que el rol del nuevo miembro es gestor.
	3a-b. El sistema proporciona permisos de gestor al nuevo miembro y el caso de uso continúa con la secuencia normal.
	3b-a. El gestor solicita la cancelación del registro
	3b-b. El sistema cancela el proceso y el caso de uso termina.
Postcondición	El nuevo miembro y sus datos han sido registrados en el sistema.

Tabla 3: Caso de uso: Registro de un nuevo miembro

UC-02	**Eliminación de un miembro de la organización**
Descripción	El gestor elimina un miembro de la organización.
Precondición	El gestor conoce el identificador del miembro que se desea eliminar, y el miembro de la organización no se ha eliminado de la organización.
Secuencia normal	1. El gestor solicita la eliminación de un miembro perteneciente a la organización.
	2. El sistema solicita el identificador del miembro que se desea eliminar.
	3. El gestor proporciona el identificador del miembro a eliminar.
	4. El sistema localiza las comunidades a las que pertenece el miembro.
	5. El sistema aplica el UC-03 para cada una de las comunidades localizadas, siendo dado de baja de todas las comunidades.
	6. El sistema elimina el registro del miembro.
Flujos alternativos	3a. El gestor solicita la cancelación de la eliminación.
	3b. El sistema cancela el proceso y el caso de uso termina.
Postcondición	Los datos y certificados relacionados con el miembro han sido eliminados.

Tabla 4: Caso de uso: Eliminar un miembro

UC-03	Dar de baja un miembro de una comunidad
Descripción	El gestor elimina un miembro de una comunidad de la organización.
Precondición	El gestor conoce el identificador del miembro que se desea eliminar y el nombre de la comunidad de la cual va a ser eliminado.
Secuencia normal	1. El gestor solicita la eliminación de un miembro perteneciente a una comunidad de la organización.
	2. El sistema solicita el identificador del miembro que se desea eliminar y el nombre de la comunidad de la cual va a ser eliminado.
	3. El gestor proporciona los identificadores solicitados por el sistema.
	4. El sistema localiza todos los certificados del miembro para esa comunidad.
	5. El sistema aplica el UC-06.
	6. El sistema elimina todos los certificados localizados de la descripción actual del sistema.
Flujos alternativos	3a. El gestor solicita la cancelación de la eliminación.
	3b. El sistema cancela el proceso y el caso de uso termina.
Postcondición	Los certificados que relacionan el miembro y la comunidad han sido eliminados.

Tabla 5: Caso de uso: Eliminar un miembro

UC-04	Creación de una nueva comunidad
Descripción	El gestor crea una nueva comunidad dentro de la organización.
Precondición	El gestor conoce los datos para la creación de la comunidad y la CA raíz no se encuentra revocada.
Secuencia normal	1. El gestor solicita la creación de una nueva comunidad.
	2. El sistema solicita los datos de la comunidad.
	3. El gestor proporciona los datos de la comunidad.
	4. El sistema inicia la creación de la CA de la comunidad.
	6. El sistema genera la estructura de la CA de la comunidad .
	7. El sistema genera la solicitud de firma para la CA de la comunidad.
	8. El sistema hace que la CA Raíz firme la solicitud de la CA de la comunidad y se genera el certificado de la nueva CA.
	11. El sistema ha generado la CA y la registra en la base de datos.
	12. El sistema asocia la CA con la comunidad en la base de datos.
	13. El sistema asocia la nueva CA con la CA raíz, para determinar que ha sido la CA raíz quien la ha creado.
	13. El sistema registra la nueva comunidad.
Flujos alternativos	3a. El gestor solicita la cancelación del proceso.
	3b. El sistema cancela el proceso y el caso de uso termina.
Postcondición	La comunidad y la CA intermedia han sido creadas y registradas en el sistema.

Tabla 6: Caso de uso: Creación de una nueva comunidad

UC-05	Eliminación de una comunidad
Descripción	El gestor elimina una comunidad de la organización.
Precondición	El gestor debe conocer el identificador de la comunidad a eliminar.
Secuencia normal	1. El gestor solicita la eliminación de una comunidad perteneciente a la organización.
	2. El sistema solicita el identificador de la comunidad.
	3. El gestor proporciona el identificador de la comunidad.
	4. El sistema localiza en la base de datos la comunidad y la CA asociada a ella.
	5. El sistema aplica el UC-09.
	6. El sistema elimina de los certificados emitidos por la CA de la comunidad y el certificado de la propia CA intermedia de la descripción actual del sistema.
Flujos alternativos	3a. El gestor solicita la cancelación del proceso.
	3b. El sistema cancela el proceso y el caso de uso termina.
Postcondición	La comunidad, la CA intermedia y los certificados emitidos por esta última han sido eliminados del sistema.

Tabla 7: Caso de uso: Eliminación de una comunidad

UC-06	**Revocar certificado de un miembro**
Descripción	El gestor revoca uno de los certificados de un miembro.
Precondición	El gestor debe conocer la comunidad a la que pertenece el certificado que se desea revocar.
Secuencia normal	1. El gestor solicita la revocación del certificado de un miembro.
	2. El sistema solicita el identificador del miembro y la comunidad de la cual se quiere revocar el certificado.
	3. El gestor proporciona los datos necesarios al sistema.
	4. El sistema localiza el certificado objetivo.
	5. El sistema revoca el certificado deseado.
	6. El sistema actualiza el estado del certificado de válido a revocado.
Flujos alternativos	3a. El gestor solicita la cancelación del registro.
	3b. El sistema cancela el proceso y el caso de uso termina.
	4a. El sistema no localiza ningún certificado válido y el caso de uso finaliza.
Postcondición	El certificado objetivo ha sido revocado y su estado actualizado dentro del sistema.

Tabla 8: Caso de uso: Revocar certificado de un miembro

UC-07	Regenerar el certificado de un miembro
Descripción	El gestor genera un nuevo certificado para un miembro en una comunidad.
Precondición	El gestor debe conocer el identificador del miembro que será dueño del certificado y la comunidad para la cual se identificará, el certificado que existía previamente para esa comunidad se encuentra revocado y el miembro debe pertenecer a dicha comunidad.
Secuencia normal	1. El gestor solicita la generación del certificado de un miembro.
	2. El sistema solicita el identificador del miembro y el identificador de la comunidad, para las que se desea el certificado.
	3. El gestor proporciona ambos identificadores.
	4. El sistema obtiene la información del miembro.
	5. El sistema localiza la información de la comunidad y la CA de la misma.
	6. El sistema genera el certificado con los datos anteriores.
	7. El sistema firma el certificado con la CA de la comunidad.
	8. El sistema registra el nuevo certificado en el sistema.
Flujos alternativos	3a. El gestor solicita la cancelación del proceso.
	3b. El sistema cancela el proceso y el caso de uso termina.
Postcondición	El nuevo certificado ha sido creado, el miembro añadido en la comunidad y todo ello registrado en el sistema.

Tabla 9: Caso de uso: Generar certificado de un miembro

UC-08	Introducir un miembro en una comunidad
Descripción	El gestor añade un miembro en una comunidad existente.
Precondición	El gestor debe conocer el identificador del miembro que será dueño del certificado y la comunidad en la cual se registrará.
Secuencia normal	1. El gestor solicita introducir un miembro en una comunidad.
	2. El sistema solicita el identificador del miembro y el identificador de la comunidad.
	3. El gestor proporciona ambos identificadores.
	4. El sistema localiza la información del miembro en la base de datos.
	5. El sistema localiza la información de la comunidad y la CA de la misma.
	6. El sistema genera el certificado con los datos anteriores.
	7. El sistema firma el certificado con la CA de la comunidad.
	8. El sistema registra el certificado.
	9. El sistema actualiza los datos de la comunidad donde el miembro se ha introducido.
Flujos alternativos	3a. El gestor solicita la cancelación del registro.
	3b. El sistema cancela el proceso y el caso de uso termina.
Postcondición	El miembro ha sido añadido en la comunidad y se ha actualizado en el sistema.

Tabla 10: Caso de uso: Introducir miembro en una comunidad

UC-09	**Revocar una CA.**
Descripción	El gestor revoca una CA.
Precondición	El gestor debe conocer el identificador de la CA.
Secuencia normal	1. El gestor solicita la revocación de una CA.
	2. El sistema solicita el identificador de la CA.
	3. El gestor proporciona el identificador de la CA.
	4. El sistema localiza la CA y la revoca.
	5. El sistema localiza todos los certificados emitidos por la CA y los revoca.
	6. El sistema actualiza el estado de válido a revocado de la CA y el certificado deja de estar vigente.
	7. El sistema actualiza el estado de válido a revocado de los certificados emitidos por la CA.
Flujos alternativos	3a. El gestor solicita la cancelación del registro.
	3b. El sistema cancela el proceso y el caso de uso termina.
	5a. El sistema detecta que se trata de una CA raíz y aplica el caso de uso UC-09 con cada una de las CA intermedias emitidas por la CA raíz.
Postcondición	La CA ha sido revocada y su estado actualizado en el sistema.

Tabla 11: Caso de uso: Revocar una CA comprometida

UC-10	Regenerar una CA
Descripción	El gestor regenera una CA.
Precondición	El gestor debe conocer todos los datos necesarios de la CA y la comunidad asociada, la CA intermedia debe encontrarse revocada .
Secuencia normal	1. El gestor solicita la creación de una CA.
	2. El sistema solicita los datos para la creación de la CA.
	3. El gestor aporta todos los datos solicitados.
	4. El sistema crea el certificado necesario para la CA.
	5. El sistema genera la CA a raíz del certificado anterior.
	6. El sistema actualiza la base de datos con la nueva CA creada.
Flujos alternativos	3a. El gestor solicita la cancelación del registro.
	3b. El sistema cancela el proceso y el caso de uso termina.
Postcondición	La CA ha sido regenerada y su estado actualizado en el sistema.

Tabla 12: Caso de uso: Regenerar una CA

UC-11	Modificar el rol de un miembro
Descripción	Otorgar un nuevo rol a un usuario del sistema puede implicar revocarle permisos o otorgarle unos nuevos.
Precondición	El usuario al que se le va a modificar el rol debe existir en el sistema, el gestor conoce el identificador del miembro al que se le va a modificar el rol.
Secuencia normal	1. El gestor solicita el cambio de rol de un usuario
	2. El sistema solicita el identificador del usuario y el nuevo rol que este tendrá.
	3. El gestor proporciona el identificador y el nuevo rol.
	4. El sistema localiza al usuario al que se le van a alterar los permisos.
	5. El sistema revoca todos los permisos actuales del usuario.
	6. El sistema asigna al usuario los permisos correspondientes al nuevo rol.
	7. El sistema registra el cambio de rol.
Flujos alternativos	3a. El gestor solicita la cancelación del registro.
	3b. El sistema cancela el proceso y el caso de uso termina.
Postcondición	El usuario ha obtenido los permisos correspondientes a su nuevo rol y ha perdido los que antes poseía.

Tabla 13: Caso de uso: Modificar el rol de un miembro.

UC-12	Verificar certificado
Descripción	Verificaremos la firma de un certificado.
Precondición	El certificado debe existir y debemos tener acceso al certificado de la CA intermedia que lo emitió, el gestor debe conocer el identificador del dueño y comunidad para la cual identifica dicho certificado.
Secuencia normal	1. El gestor solicita la verificación de un certificado
	2. El sistema solicita el identificador del usuario, la comunidad a la que pertenece el certificado y el certificado de la CA intermedia que sirve a dicha comunidad.
	3. El gestor proporciona todos los datos solicitados por el sistema.
	4. El sistema recoge los datos y realiza la verificación.
	5. El sistema devuelve el resultado de la verificación.
Flujos alternativos	3a. El gestor solicita la cancelación del registro.
	3b. El sistema cancela el proceso y el caso de uso termina.
Postcondición	Se ha comprobado la firma del certificado en cuestión.

Tabla 14: Caso de uso:Verificar certificado.

UC-13	Otorgar una nueva clave privada a un miembro
Descripción	Otorgamos una nueva clave a un miembro de la organización.
Precondición	La clave privada anterior del miembro ha dejado de ser utilizable debido a que ha sido expuesta, el gestor conoce el identificador del miembro al que se le va a otorgar la nueva clave.
Secuencia normal	1. El gestor solicita una nueva clave privada para un usuario del sistema.
	2. El sistema solicita el identificador del miembro al que se le otorgará la nueva clave.
	3. El gestor proporciona el identificador del usuario.
	4. El sistema genera la nueva clave del usuario.
	5. El sistema almacena la nueva clave del usuario.
	6. El sistema elimina la vieja clave del usuario.
Flujos alternativos	3a. El gestor solicita la cancelación del registro.
	3b. El sistema cancela el proceso y el caso de uso termina.
Postcondición	El usuario ha obtenido una nueva clave y ha perdido la que poseía anteriormente, el sistema ha registrado estos cambios.

Tabla 15: Caso de uso:Otorgar una nueva clave privada a un miembro.

Capítulo 5 - Diseño

En este capítulo veremos detallado el estudio realizado para realizar el diseño del proyecto. Las explicaciones que podremos ver aquí se verán reforzadas mediante el uso de diagramas que puedan permitir mostrar de manera gráfica el sistema que nos encontramos desarrollando y la situación o situaciones donde este es aplicable.

1. Descripción del entorno

Para comenzar describiremos el escenario frente al que nos encontramos. Durante las secciones anteriores hemos hablado sobre un gran grupo que se encontraba subdividido en subgrupos de menor tamaño que tienen diferentes miembros que a su vez pueden encontrarse en varias de estos subgrupos de manera simultánea, el siguiente diagrama muestra este escenario.

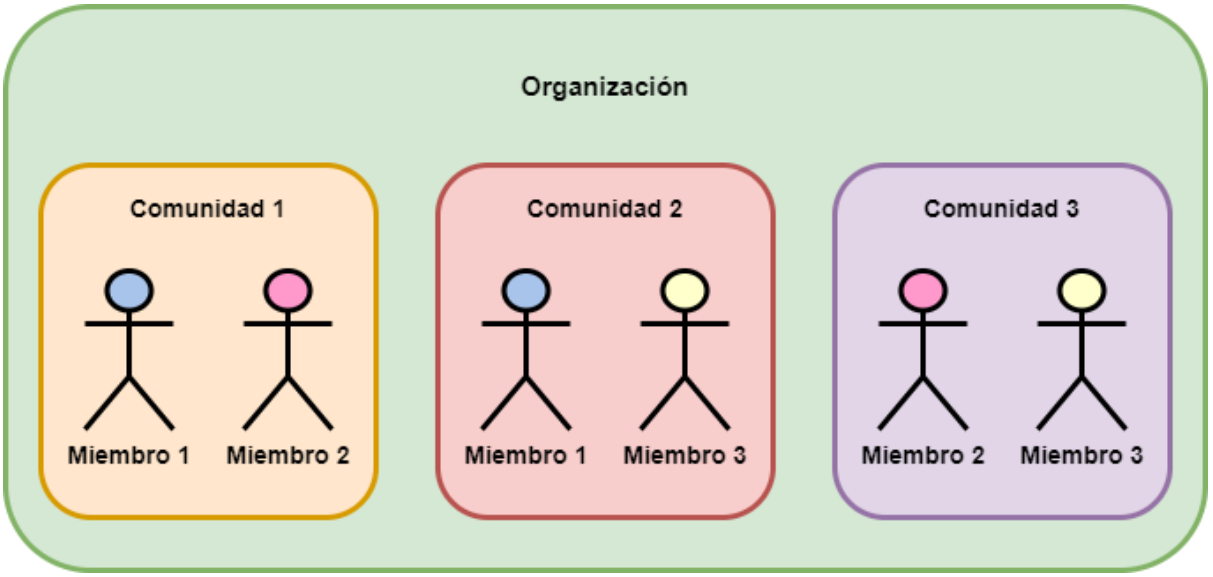


Figura 6: Escenario para aplicar el sistema

Para poder lograr la condición de que un mismo miembro pueda identificarse como miembro de varias comunidades este mismo escenario que hemos mostrado podemos replicarlo mediante una estructura de CAs (autoridades certificadoras) y certificados digitales, ¿Pero por qué debemos replicar esta estructura?. Como hemos explicado durante la fundamentación teórica no pueden coexistir dos certificados digitales iguales, la generación del segundo invalidará de manera automática el primero existente, pero replicando el escenario anterior empleando la estructura de CAs y certificados digitales podremos lograr cumplir ambas condiciones. Un miembro podría ser poseedor de varios certificados sin la necesidad de que estos sean iguales y de esta forma ambos se mantendrán válidos para certificar la pertenencia a las diferentes comunidades.

El diagrama que mostramos a continuación muestra la replicación del escenario usando la estructura de CAs y certificados.

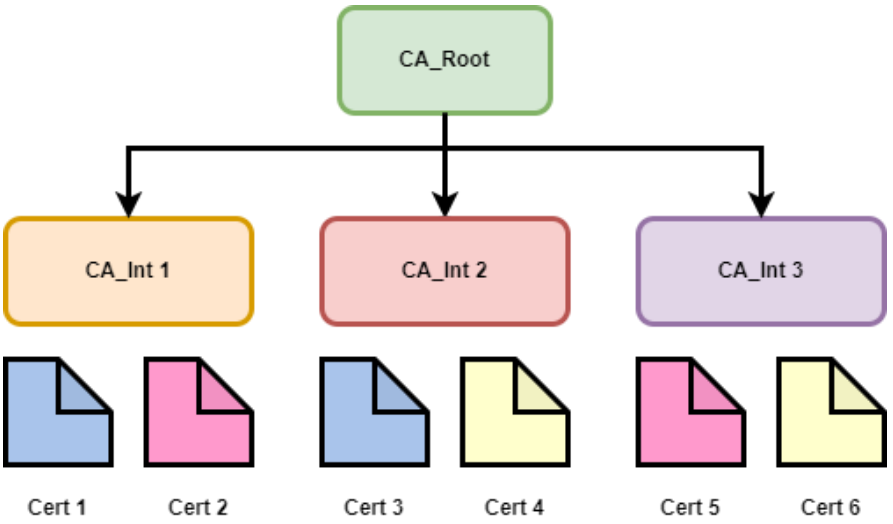


Figura 7: escenario replicado mediante estructura de CAs y certificados

Como podemos ver mantenemos la jerarquía mostrada en el diagrama anterior pero trasladado a un sistema de PKI; no debemos de olvidar esta jerarquía de la cual ya hemos hablado en la fundamentación teórica, en una situación normal y en nuestro caso la existencia de la CA_Root (Autoridad Certificadora Raíz) se mantendría del mismo modo, pero es las CA_Int (Autoridad Certificadora Intermedia) donde encontraremos el músculo de nuestro sistema, ya que en condiciones normales solo debería existir una, pero nuestro sistema podrá generar tantas como comunidades existan en la organización de manera que cada comunidad dispondrá de su propia CA_Int que permita la emisión de los certificados digitales para que los miembros de estas puedan identificarse de manera adecuada. Al tratar de diferentes CAs un mismo miembro podrá disponer de más de un certificado, pero sin perder la posibilidad de acreditar que las comunidades a las que éste pertenece se encuentran dentro de la misma organización debido a que todas las CA_Int de la organización han sido creadas por la misma CA_Root por lo que existe una cadena de certificados que aseguran esto.

Entendemos la cadena de certificados como aquella que va desde el certificado que un miembro posee hasta el certificado propio de la CA_Root pasando por la CA_Int.

La estructura jerárquica que hemos mostrado a su vez permitiría aplicarse de manera recursiva dentro de una comunidad que pudiera encontrarse subdividida también, siendo capaz de mantener esa cadena de caracteres.

2. Modelo de datos

En esta sección presentaremos el modelo de datos que nos ayude a presentar el sistema que en las próximas secciones vamos a diseñar e implementar.

Este modelo como hemos comentado anteriormente será el encargado de describir la situación de la organización y nos permitirá observar cuáles son los elementos encargados de identificar a nuestros miembros dentro de la organización. En primer lugar mostraremos el diagrama y seguidamente lo explicaremos.

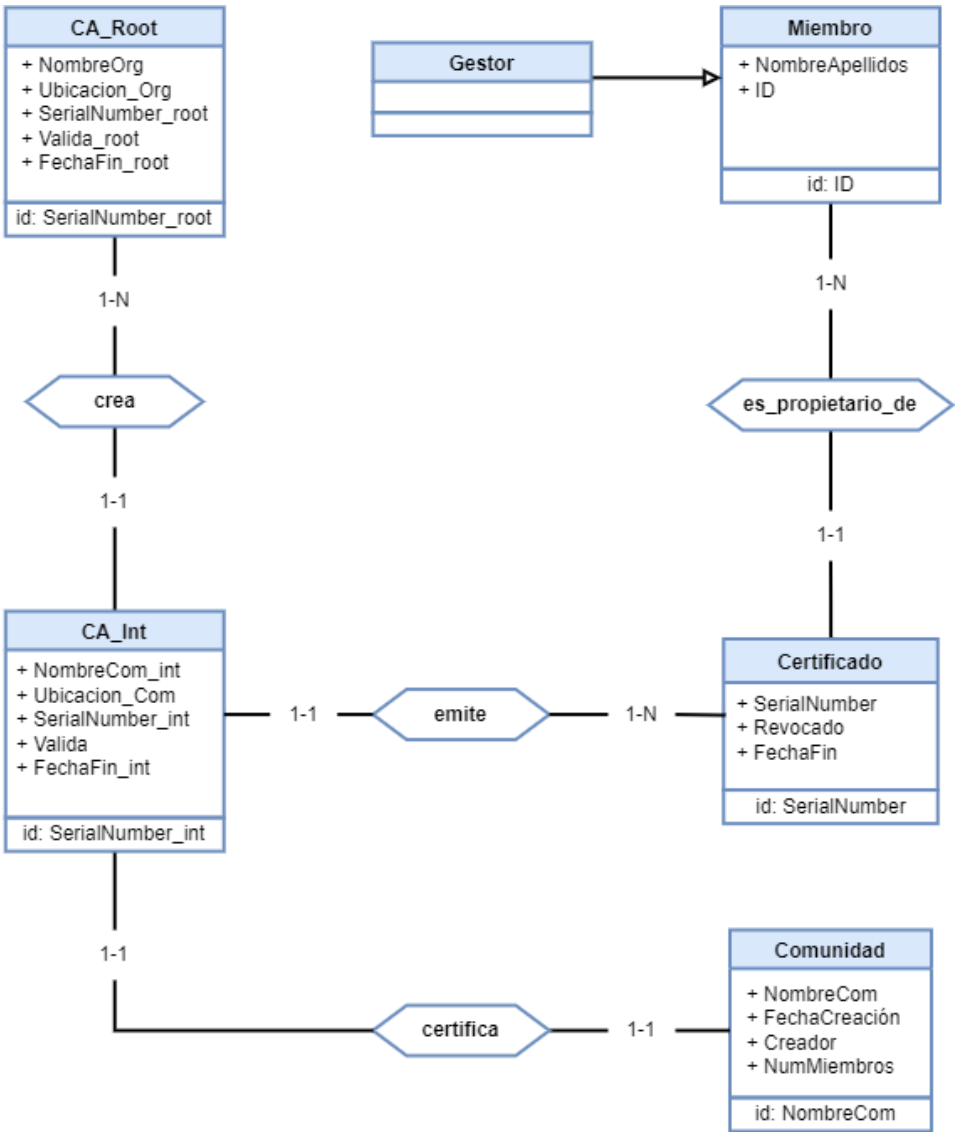


Figura 8: Modelo de datos

Con el diagrama ya presentado ahora nos será mucho más sencillo explicarlo ya que sin tener este claro será complicado seguir los pasos siguientes. Como podemos ver cumpliendo con los requisitos de privacidad almacenaremos únicamente los datos necesarios para generar los certificados que cada miembro de la organización necesite para acreditar su pertenencia a las diferentes comunidades de las que este es miembro, así como los datos que permitan la creación de las diferentes CAs (autoridades certificadoras) que existirán en nuestro sistema. Tenemos que destacar que aunque el miembro no tenga el certificado no quiere decir que este no sea miembro de la comunidad, únicamente no podrá identificarse como miembro de la misma. Para entenderlo usaremos un ejemplo: es como un policía sin su placa, este continúa siendo policía, pero sin su placa no puede demostrarlo.

Lo más importante que nos muestra el diagrama es la posibilidad de observar que un único miembro tenga en su posesión diferentes certificados que le permitan identificarse como perteneciente a una comunidad.

En la parte izquierda del diagrama podemos apreciar la jerarquía de CAs que permiten la emisión de certificados para las distintas comunidades. Podemos observar la CA_Root que será única y la encargada de soportar el sistema que estamos construyendo, mediante la creación de las CA_Int que serán las que posibiliten la característica de la que hemos hablado en el párrafo anterior.

Podemos observar que los atributos de la entidad certificado, son bastante reducidos, esto se debe a que los atributos restantes podemos obtenerlos de la cadena de relaciones que posee esta entidad.

También como hemos presentado en los requisitos podemos ver la existencia de gestor como una especialización de un miembro estándar ya que estos a pesar ser miembros normales en el sentido de pertenencia a una o varias comunidades estos son los únicos habilitados para gestionar el sistema.

2.1 Descripción de las entidades y atributos del modelo

En las siguientes tablas describiremos cada una de las entidades del diagrama mostrado anteriormente así como cada atributo que componen estas, de manera que la descripción que hemos hecho con anterioridad pueda ser más clara.

Certificado		
Archivo que permite la identificación de un miembro de una comunidad.		
SerialNumber	Número de identificación del certificado es único entre todos los certificados dentro del sistema, incluidos los certificados que han sido revocados.	Int
Revocado	Atributo que indica el estado del certificado. - False: válido True: revocado	Boolean
FechaFin	Fecha hasta la cual es válido el certificado.	Date

Tabla 3: Descripción de los elementos de "Certificado"

CA_Int		
Autoridad certificadora intermedia, encargada de la emisión de certificados para los diferentes miembros de la organización.		
NombreCom_Int	Nombre de la comunidad a la que pertenece dicha CA_Int.	VarChar
Ubicacion_Com	Ubicación donde se encuentra la comunidad, no tiene por que ser la misma que la ubicación de la organización.	Varchar
SerialNumber_Int	Número de identificación de la CA, es de único y tiene caracter identificativo.	Int
Valida	Atributo que indica el estado de la CA - False: válida True: revocado	Boolean
FechaFin_Int	Fecha hasta la cual es válido el certificado.	Date

Tabla 4: Descripción de los elementos de "CA_Int"

CA_Root		
Autoridad certificadora raíz, encargada de la creación de las CA_Int para las diferentes comunidades.		
NombreOrg	Nombre de la organización a la que pertenece dicha CA_Root.	VarChar
Ubicacion_Org	Ubicación principal donde se encuentra la organización.	Varchar
SerialNumber_root	Número de identificación de la CA, es de único y tiene caracter identificativo.	Int
Valida_root	Atributo que indica el estado de la CA. - False: válida True: revocado	Boolean
FechaFin_root	Fecha hasta la cual es válido el certificado.	Date

Tabla 5: Descripción de los elementos de "CA_Root"

Comunidad		
Subgrupo de personas y medios organizados con un fin determinado dentro de una organización.		
NombreCom	Nombre de la comunidad.	VarChar
FechaCreacion	Fecha en la que se creó la comunidad.	Date
Creador	Gestor que ha creado la comunidad.	VarChar
NumMiembros	Número de miembros que forman la comunidad	Int

Tabla 6: Descripción de los elementos de "Comunidad"

Miembro		
Persona perteneciente a la organización y los cuales se encargan de componer las diversas comunidades.		
NombreApellidos	Nombre y apellidos de la persona.	VarChar
ID	Código de identificación del miembro dentro de su organización	VarChar

Tabla 7: Descripción de los elementos de "Miembro"

Gestor
Persona perteneciente a la organización, es una especialización de "Miembro", ya que estos son los encargados de gestionar el sistema ya que esta es una función reservada solo para algunos.

Tabla 8: Descripción de los elementos de "Gestor"

La necesidad de almacenar el atributo ID surge de la necesidad de identificar de manera única a cada miembro y gestor de la organización, este problema ya no existe "dentro" de los certificados ya que para ello emplea la clave pública pero a la hora de facilitar el tratamiento de datos por parte de los gestores del sistema facilita mucho esta tarea.

2.2 Descripción de las relaciones del modelo

Como podemos ver entre los elementos del modelo existe una relación, que ahora pasaremos a detallar para terminar de explicar el diagrama que hemos presentado anteriormente. Las relaciones que hemos podido identificar son:

- **es_propietario_de:** Esta relación determina cuales son los certificados de los cuales es propietario un miembro de la organización.

- **certifica:** Esta relación determina qué CA certificados los certificados de una determinada comunidad.

- **emite:** Esta relación determina que CA_Int ha sido la encargada de generar ciertos certificados.

- **crea:** Esta relación nos ayuda a mostrar la jerarquía entre las diferentes CAs determinando que CA es la creadora de otra.

3. Diseño de la base de datos

Tras haber estudiado el modelo de datos del sistema que nos encontramos desarrollando, continuaremos por mostrar cómo serán las diferentes tablas que compondrán la base de datos encargada de facilitar la gestión de nuestro sistema tras la implementación dentro de una organización.

3.1 Transformación de las entidades del modelo en tablas

Ahora explicaremos la transformación de las entidades presentadas en el modelo de datos (**figura 7**) en las respectivas tablas que estarán presentes en nuestra base de datos.

En cuanto a las tablas que se generen derivadas de las entidades del modelo no variarán en gran media de las propias entidades quedando dichas tablas de la siguiente manera, tratándose de el o los atributos subrayados el identificador, especificaremos tambien las claves foráneas las cuales son atributos que deben existir previamente en otras tablas para poder ser insertados, de manera que se mantenga la integridad:

- **MIEMBRO** (NombreApellidos, ID)

- **GESTOR** (NombreApellidos, ID)

- **CERTIFICADO** (SerialNumber, Revocado, FechaFin)

- **COMUNIDAD** (NombreCom, FechaCreacion, Creador, NumMiembros)
    - Clave Foránea: Creador con GESTOR (ID)

- **CA_INT** (NombreCom_int, Ubicación_Com, SerialNumber_int, Valida, FechaFin_int)
    - Clave Foránea: NombreCom_int con COMUNIDAD (NombreCom)

- **CA_ROOT** (NombreOrg_cert, Ubicacion_Org, SerialNumber_root, Valida_root, FechaFin_root)

  Aunque verdaderamente esta tabla únicamente solo vaya a tener una entrada como hemos podido ver en la descripción del escenario, resulta muy interesante almacenar los datos de la CA_Root, de cara a un futuro mantenimiento del sistema, puesto que que una CA no tiene una validez infinita y ya que podría verse expuesta y ser

necesaria una recuperación total del sistema, tarea la cual se vería facilitada si conservamos esta información sobre CA_Root.

A diferencia de las tablas que hemos visto que se han generado con las entidades de las relaciones también surgirán tablas pero estas lo harán de una manera distinta ya que estas no se "autoconstruyen" como ha ocurrido con las tablas nacidas de las entidades.

3.2 Transformación de las relaciones del modelo en tablas

Ahora explicaremos la transformación de las relaciones presentadas en el modelo de datos (**figura 7**) en las respectivas tablas que estarán presentes en nuestra base de datos.

Las tablas nacidas de las relaciones del modelo se construyen gracias a los identificadores de las entidades que estás relacionan, además los datos contenidos estas tablas se verán limitados por las cardinalidades del modelo que se han presentado en el modelo. En primer lugar presentaremos todos los componentes en una tabla y seguidamente los trasladaremos a las tablas finales que nazcan de estas relaciones.

Relación	**Entidades participantes**	**Cardinalidades**
es_propietario_de	Miembro (id: ID) Certificado (id: SerialNumber)	card_min(Miembro, ID) = 1 card_max(Miembro, ID) = N card_min(Certificado,SerialNumber) = 1 card_max(Certificado,SerialNumber) = 1
certifica	Certificado (id: SerialNumber_int) Comunidad (id: NombreCom)	card_min(CA_Int,SerialNumber_int) = 1 card_max(CA_Int,SerialNumber_int) = 1 card_min(Comunidad,NombreCom) = 1 card_max(Comunidad,NombreCom) = 1
emite	CA_Int (id: SerialNumber_int) Certificado (id: SerialNumber)	card_min(CA_Int,SerialNumber_int) = 1 card_max(CA_Int,SerialNumber_int) = N card_min(Certificado,SerialNumber) = 1 card_max(Certificado,SerialNumber) = 1
crea	CA_Root (id: SerialNumber_root) CA_Int (id: SerialNumber_int)	card_min(CA_Root,SerialNumber_root) = 1 card_max(CA_Root,SerialNumber_root) = N card_min(CA_Int,SerialNumber_int) = 1 card_max(CA_Int,SerialNumber_int) = 1

Tabla 9: Componentes de las tablas de relación

Con los componentes de las tablas de relación presentados de manera adecuada podemos definir ya cuáles serán estas tablas. Es importante destacar el valor de estas tablas ya que sin

ellas el tamaño de ciertas tablas de entidad sería excesivamente amplio como para ser capaces de realizar un uso óptimo de las mismas, si bien es cierto que estas tablas de relación pueden tener muchas líneas, la cantidad reducida de atributos que estas poseen hace que se mucho más simple su uso y de esta forma evitar problemas derivados de tablas con gran cantidad de líneas y atributos.

Las tablas de relación que existirán en nuestro sistema serán las siguientes, al igual que hemos realizado anteriormente el atributo subrayado se trata del identificador, que en este caso surgirá de las cardinalidades que hemos presentado en la tabla anterior, también mostraremos las claves foráneas de estas tablas ya que serán un elemento clave para mantener la integridad de nuestra base de datos:

- **es_propietario_de** (ID, SerialNumber)
    - Clave Foránea: ID con MIEMBRO (ID).
    - Clave Foránea: SerialNumber con CERTIFICADO (SerialNumber).

- **certifica** (SerialNumber_int, NombreCom)
    - Clave Foránea: NombreCom con COMUNIDAD (NombreCom).
    - Clave Foránea: SerialNumber_int con CA_INT (SerialNumber_int).

- **emite** (SerialNumber_int, SerialNumber)
    - Clave Foránea: SerialNumber_int con CA_INT (SerialNumber_int).
    - Clave Foránea: SerialNumber con CERTIFICADO (SerialNumber).

- **crea** (SerialNumber_root, SerialNumber_int)
    - Clave Foránea: SerialNumber_int con CA_INT (SerialNumber_int).
    - Clave Foránea: SerialNumber_root con CA_ROOT (SerialNumber_root).

3.3 Roles en la base de datos

En último lugar deberemos analizar quiénes serán los usuarios de esta base de datos y de qué manera estos interactúan con ella por lo que existirán roles.

Durante nuestro análisis hemos podido identificar dos grupos de usuarios en nuestro sistema, los miembros y los gestores, a cada uno de estos grupos les corresponderá un rol . Así que a continuación describiremos qué clase de permisos deberían de tener estos roles sobre las diferentes tablas que compondrán nuestra base de datos, para ello haremos uso de una matriz de accesos que nos permitirá mostrarlo de una manera gráfica y sencilla.

	Usuarios	
Tablas	Gestor	Miembro
MIEMBRO	select, insert, update, delete	select
GESTOR	select, insert, update, delete	select
COMUNIDAD	select, insert, update, delete	select
CERTIFICADO	select, insert, update, delete	X
CA_INT	select, insert, update, delete	X
CA_ROOT	select, insert, update, delete	X
es_propietario_de	select, insert, update, delete	X
certifica	select, insert, update, delete	X
emite	select, insert, update, delete	X
crea	select, insert, update, delete	X

Tabla 10: Matriz de accesos

A la vista de la matriz que hemos construido vemos claramente cuales son las diferencias que existen entre los dos roles que hemos establecido, mientras que los gestores podrán realizar cualquier acción, que no altere el esquema de la base de datos, mientras que los miembros solo podrán consultar información información de aquellas tablas que no contienen información sensible o que pueda alterar el funcionamiento del sistema.

Capítulo 6 - Implementación y validación

Durante el desarrollo de este capítulo explicaremos cómo hemos realizado la implementación y validación de nuestro sistema, detallando las partes que este contiene las tecnologías empleadas y por que a día de hoy consideramos que son las elecciones correctas para implementar el sistema que hemos desarrollado.

Como hemos explicado anteriormente el sistema que nos encontramos desarrollando busca poder adaptarse a cualquier organización y por ello realizaremos dicha implementación sobre un escenario ficticio de las mismas características que las que se presentan en la **figura 5**. Nos veremos ante una organización compuesta por tres comunidades (Comunicaciones, Seguridad y Privacidad) y por tres miembros (Alice, Bob y Mallet).

1. Descripción de la máquina de soporte

Antes de comenzar a dar detalles sobre cómo hemos implementado nuestro sistema debemos analizar qué tipo de máquina será la encargada de soportar nuestro sistema .

La máquina escogida se trata de un máquina dotada con 16 GB de memoria RAM y 50 GB de disco duro, también debemos destacar que el sistema operativo instalado no es otro que Ubuntu 20.04, el cual se trata de una distribución de tipo Linux.

Debemos de destacar el hecho de usar una distribución de tipo Linux y no Windows, y es que esta decisión ha sido tomada en cuanto que la compatibilidad de ciertas herramientas es mucho mayor con Linux que con Windows, ya que las normalmente diseñadas para este último sistema operativo son mucho más complejas y no permiten sencillez a la hora de interconectar diversas herramientas, otro punto a favor de la elección de Linux ha residido en los costes del proyecto ya que al apostar por software libre y de código abierto la implementación de nuestro sistema será mucho más económica.

Seguidamente pasaremos a describir cómo son las partes del sistema que hemos desarrollado, el cual podemos verlo dividido en dos partes o elementos diferentes los cuales deben actuar en conjunto para no crear conflictos. Estos dos elementos los conoceremos como “Esquema de certificados” y “Base de Datos”. A continuación detallaremos cada una de estas partes tanto la estructura que estos poseen como las tecnologías y configuraciones que actualmente consideramos como aptas para el uso del sistema en una organización.

2. Esquema de certificados

Esta es la parte principal de nuestro sistema, podemos considerarlo el núcleo, será el elemento encargado de lograr la emisión de los diferentes certificados que los miembros de la organización soliciten. Además esta parte del sistema es la encargada de almacenar los diferentes certificados que se emiten y han sido emitidos, para en caso de ser necesaria una auditoría disponer de todos certificados digitales en uso o obsoletos.

Esta parte de nuestro sistema se encontrará formada por las diferentes autoridades certificadoras (CAs) y certificados digitales, siguiendo la estructura que pudimos presentar en el capítulo anterior de diseño.

2.1 Tecnología empleada

Para la selección de la herramienta tecnológica, que hemos empleado para la construcción de este elemento, hemos observado principalmente factores como compatibilidad, recursos necesarios, estabilidad, sencillez y adaptabilidad. Por ello la herramienta escogida no ha sido otra que OpenSSL, la cual se trata de una biblioteca de software de criptografía. **[11]**

La versión que hemos empleado de OpenSSL ha sido la 1.1.1 que si bien no es la más moderna el la más utilizada a día de hoy en distribuciones como la nuestra. Hemos evitado el uso de las versiones anteriores a la 1.1.1 puesto que poseían un agujero de seguridad conocido como *Heartbleed* [3] .

OpenSSL contiene gran cantidad de herramientas y por lo tanto configuraciones las cuales detallaremos en el próximo apartado.

2.2 Configuraciones aplicadas

La configuración de OpenSSL reside en el fichero *openssl.cnf* por lo que será este el que debamos alterar para modificar ciertas opciones aunque otras simplemente será necesario emplear la opción correcta en la ejecución de la herramienta.

La principal configuración que alteramos en el fichero *openssl.cnf* son valores por defecto para la generación de los distintos certificados así como las rutas en las que las diferentes herramientas necesitan localizar algún archivo, pero también hemos realizado la construcción de una nueva extensión para que esta pueda ser aplicada a las CAs intermedias, las modificaciones comentadas las podemos observar destacadas en las siguientes imágenes. Debemos comentar de que la existencia de múltiples CAs también deriva en la existencia de

[3] *Heartbleed* se trata del agujero de seguridad detectado en la versión 1.0.1 de OpenSSL el cual permitía al atacante leer la memoria de un cliente o servidor permitiendo conseguir las claves privadas alojadas. **[17] [19]**

múltiples archivos de configuración ya que diferentes CAs necesitan diferentes archivos y es necesario que esto se vea reflejado en el *openssl.cnf* correspondiente.

```
####################################################################
[ CA_default ]

dir             = /DigIdCom/ca                  # Where everything is kept
certs           = $dir/certs            # Where the issued certs are kept
crl_dir         = $dir/crl              # Where the issued crl are kept
database        = $dir/index.txt        # database index file.
#unique_subject = no                    # Set to 'no' to allow creation of
                                        # several certs with same subject.
new_certs_dir   = $dir/newcerts         # default place for new certs.

certificate     = $dir/certs/ca_root.cert.pem   # The CA certificate
serial          = $dir/serial           # The current serial number
crlnumber       = $dir/crlnumber        # the current crl number
                                        # must be commented out to leave a V1 CRL
crl             = $dir/crl.pem          # The current CRL
private_key     = $dir/private/ca_root.key.pem# The private key
```

Figura 9: Rutas archivo openssl.cnf CA raíz

Esta captura corresponde al fichero *openssl.cnf* de la CA raíz y en él se han alterado las rutas de *dir, certificate y private_key.*

Para cada una de las CAs intermedias la modificación es similar y la podemos observar en la siguiente figura.

```
####################################################################
[ CA_default ]

dir             = /DigIdCom/ca/intermediate$COMMUNITY   # Where everything is kept
certs           = $dir/certs            # Where the issued certs are kept
crl_dir         = $dir/crl              # Where the issued crl are kept
database        = $dir/index.txt        # database index file.
#unique_subject = no                    # Set to 'no' to allow creation of
                                        # several certs with same subject.
new_certs_dir   = $dir/newcerts         # default place for new certs.

certificate     = $dir/certs/inter$COMMUNITY.cert.pem # The CA certificate
serial          = $dir/serial            # The current serial number
crlnumber       = $dir/crlnumber         # the current crl number
                                         # must be commented out to leave a V1 CRL
crl             = $dir/crl.pem           # The current CRL
private_key     = $dir/private/inter$COMMUNITY.key.pem # The private key
```

Figura 10: Rutas archivo openssl.cnf CAs intermedias

Podemos observar la existencia del path *$COMMUNITY* el cual permite alterar la comunidad para la que dicho archivo sirve únicamente cambiando el valor de este path que se encuentra en la parte superior del archivo.

```
COMMUNITY=Comunicaciones

#
# OpenSSL example configuration file.
# This is mostly being used for generation of certificate requests.
#
```

Figura 11: Path de los ficheros de configuración de las CAs intermedias

Las configuraciones realizadas de valores por defecto para rellenar los campos de los certificados quedarían de la siguiente manera.

```
[ req_distinguished_name ]
countryName                     = Country Name (2 letter code)
countryName_default             = ES
countryName_min                 = 2
countryName_max                 = 2

stateOrProvinceName             = State or Province Name (full name)
stateOrProvinceName_default     = Castilla_y_Leon

localityName                    = Locality Name (eg, city)
localityName_default            = Valladolid

0.organizationName              = Organization Name (eg, company)
0.organizationName_default      = DigIdCom

# we can do this but it is not needed normally :-)
#1.organizationName             = Second Organization Name (eg, company)
#1.organizationName_default     = World Wide Web Pty Ltd

organizationalUnitName          = Organizational Unit Name (eg, section)
organizationalUnitName_default  = $COMUNITY

commonName                      = Common Name (e.g. server FQDN or YOUR name)
commonName_max                  = 64
```

Figura 12: Valores por defecto de los campos del certificado en el fichero openssl.cnf

Podemos ver que aquí volvemos a hacer uso del path para que ese campo se vuelva por defecto el de la comunidad a la cual dará servicio la CA intermedia. En el fichero de configuración de la CA raíz este campo en lugar del path tendrá el valor por defecto de “DigIdCom”.

La última configuración que hemos alterado ha sido la política de CA para obligar a que la unidad organizativa (comunidad) del certificado que es emitido por la CA intermedia sea la misma que la de la CA intermedia.

```
# A few difference way of specifying how similar the request should look
# For type CA, the listed attributes must be the same, and the optional
# and supplied fields are just that :-)
policy          = policy_match

# For the CA policy
[ policy_match ]
countryName             = match
stateOrProvinceName     = match
organizationName        = match
organizationalUnitName  = match
commonName              = supplied
emailAddress            = optional
```

Figura 13: Política de la CA en los ficheros de configuración de las CAs intermedias

Esta configuración solo la hemos realizado en los archivos de configuración de las CAs intermedias ya que de hacerlo en el archivo de la CA raíz no podríamos emitir certificados con distintas unidades organizativas, que es lo que queremos para nuestras CAs intermedias.

La última configuración que hemos necesitado alterar dentro de los ficheros de configuración ha sido la versión de TLS. TLS o Transport Layer Security (seguridad de la capa de transporte) no se trata más que de la versión actualizada del protocolo SSL o Secure Socket Layer (capa de puertos seguros) el cual nos permite que la información transmitida por una red no pueda ser interceptada ni modificada por elementos no autorizados. Puesto que OpenSSL nos permite usar la versión más reciente de este protocolo y al tratarse de un protocolo que proporciona seguridad pues será esta versión la que emplearemos, TLSv1.3.

Existen otra infinidad de configuraciones dentro de estos archivos pero no ha sido necesario alterarlas ya que podemos hacerlo haciendo uso de las opciones de las herramientas de manera que es mucho más sencillo y claro. Principalmente las dos configuraciones que hemos alterado no son otras que la longitud de bits de las claves y el cifrado que emplearemos. Estas configuraciones quedarían de la siguiente manera:

- **Longitud de bits de las claves: 2048**

  Habitualmente la cantidad empleada era 1024, pero esta cantidad de bits en los tiempos actuales se han visto convertidos en vulnerables por equipos con suficiente capacidad computacional, por lo que el siguiente paso para la longitud de la clave sería el siguiente paso exponencial.

$$2^{10} = 1024 \quad \rightarrow \quad 2^{11} = 2048$$

- **Función hash empleada:** **SHA256**

  A la hora de seleccionar la función de resumen hash que emplearemos lo importante e indispensable será escoger aquella que menos vulnerable sea. Esta búsqueda de lograr una menor vulnerabilidad hace que automáticamente debamos descartar funciones de hash como MD5 y SHA-1[4], teniendo que apostar por SHA-2 el cual se encuentra dividido en dos versiones SHA256 y SHA512, la diferencia entre ambas versiones es el tamaño de palabra que emplean, 32 bytes y 64 bytes respectivamente. Pero el uso de SHA512 no es algo muy extendido por lo que podría ser contraproducente emplear ya que podría generar incompatibilidades si quisiéramos integrar nuestro sistema con otro. Por lo que optamos por el uso de SHA256.

2.3 Construcción

Para generar nuestro Esquema de certificados y que ciertos procesos que realizaremos sean replicables haremos uso de ciertos archivos scripts que explicaremos a continuación mientras hacemos uso de ellos para construir esta parte de nuestro sistema.

Como hemos explicado anteriormente deberemos comenzar por la creación de nuestra autoridad certificadora raíz (CA_Root). Para ello haremos uso del script *CA_Root.sh* el cual se muestra en la siguiente figura.

```
CA_PATH=CA_Root

mkdir $CA_PATH $CA_PATH/certs $CA_PATH/crl $CA_PATH/newcerts $CA_PATH/private $CA_PATH/privateMember

cd $CA_PATH
cp /DigIdCom/openssl.cnf .

chmod 700 private
chmod 700 privateMember
touch index.txt
echo 0001 > serial
echo 0001 > crlnumber

#################################################################################################################

openssl genrsa -out private/ca_root.key.pem 2048        #llave privada
chmod 400 private/ca_root.key.pem

#################################################################################################################

openssl req -config openssl.cnf -key private/ca_root.key.pem -new -sha256 -x509 -days 5000 -extensions v3_ca \
        -subj "/C=ES/ST=Castilla_y_Leon/L=Valladolid/O=DigIdCom/OU=DigIdCom/CN=DigIdCom_RootCA" \
        -out certs/ca_root.cert.pem
```

Figura 14: Script CA_Root.sh

Como podemos observar este script se divide en tres partes. La primera de ellas será la encargada de crear la estructura de la CA_Root, creando los directorios necesarios para el almacenamiento de los certificados y las solicitudes de revocación (crl), al tratarse de una CA

[4] Desde el año 2004 y 2005 MD5 y SHA-1 se han visto comprometidos de manera matemática; y en el año 2017 un equipo formado por Google y CWI Amsterdam logró encontrar la primera colisión de SHA-1, algo lo cual debería ser imposible ya que la misión de una función hash es lograr una huella digital única, cuando existen dos huellas digitales iguales existe una colisión. **[7] [8]**

raíz no será necesario una ubicación de solicitudes de firmas (csr) debido a que esta es autofirmada por sí misma, también observamos que se configuran los permisos para los directorios *private* y *privateMember* ya que en estos se almacenan los certificados de la CA_Root y los certificados de los miembros del sistema respectivamente.

La segunda parte será la encargada de generar la clave privada de la CA con una longitud de 2048 bits esto lo haremos haciendo uso de la función *genrsa*, posteriormente cuando la clave esta generado y almacenada alteramos los permisos de esta para que únicamente el gestor del sistema pueda leerla y únicamente leerla ya que no necesita verse empleada de otra manera.

La tercera y última parte sería la encargada de generar la solicitud de firma mediante la función *req,* pero al tratarse de un certificado autofirmado podremos obtener directamente el certificado de nuestra CA_Root.

Tras la ejecución de este script obtenemos la siguiente estructura.

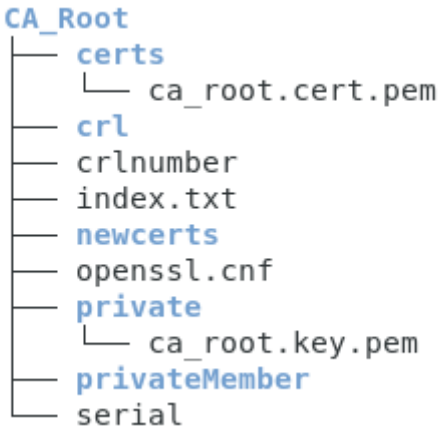


Figura 15: Estructura de Certificados tras ejecuitar CA_Root.sh

Podemos observar la existencia de un fichero denominado index.txt el cual es el registro de los certificados que esa CA ha emitido y el estado en el que se encuentran. Para nosotros nos será de utilidad de cara a una futura auditoría ya que nos permitirá conocer todos los certificados que han existido y a quienes han pertenecido.

El siguiente paso será la creación de las autoridades certificadoras intermedias (CA_intermediate) lo cual lo haremos con el script *CA_intermediate.sh* el cual tiene la siguiente estructura.

```
COMUNITY_PATH=$1
serialPath=$2

#NO TOUCH
INTER_PATH=intermediate$COMUNITY_PATH
INTERKEY_PATH=inter$COMUNITY_PATH.key.pem
INTERCSR_PATH=inter$COMUNITY_PATH.csr.pem
INTERCERT_PATH=inter$COMUNITY_PATH.cert.pem
INTERCONF_PATH=openssl$COMUNITY_PATH.cnf
INTERVALUES_PATH="/C=ES/ST=Castilla_y_Leon/L=Valladolid/O=DigIdCom/OU=$COMUNITY_PATH/CN=DigIdCom_IntermediateCA_
$COMUNITY_PATH"

cd CA_Root

mkdir -p $INTER_PATH $INTER_PATH/certs $INTER_PATH/crl $INTER_PATH/csr $INTER_PATH/newcerts $INTER_PATH/private
chmod 700 $INTER_PATH/private
touch $INTER_PATH/index.txt
echo $serialPath > $INTER_PATH/serial
echo $serialPath > $INTER_PATH/crlnumber
#########################################################################################################
openssl genrsa -out $INTER_PATH/private/$INTERKEY_PATH 2048
chmod 400 $INTER_PATH/private/$INTERKEY_PATH
#########################################################################################################
cp /DigIdCom/$INTERCONF_PATH /DigIdCom/CA_Root/$INTER_PATH

openssl req -config $INTER_PATH/$INTERCONF_PATH  -new -sha256 -key $INTER_PATH/private/$INTERKEY_PATH -subj $INT
ERVALUES_PATH -out $INTER_PATH/csr/$INTERCSR_PATH

openssl ca -config openssl.cnf -extensions v3_intermediate_ca -days 5000 -batch -notext -md sha256 -in $INTER_PA
TH/csr/$INTERCSR_PATH -out $INTER_PATH/certs/$INTERCERT_PATH

cat $INTER_PATH/certs/$INTERCERT_PATH certs/ca_root.cert.pem > $INTER_PATH/certs/chain.cert.pem

chmod 444 $INTER_PATH/certs/chain.cert.pem
```

Figura 16: Script CA_intermediate.sh

Podemos observar que la ejecución de este script necesita el paso de dos variables que son las encargadas de decidir la comunidad que se va a crear y el inicio de los números seriales para los certificados que esta CA_intermediate genere, tenemos que tener en cuenta que el gestor que emplee estos scripts debe tener un buen conocimiento del sistema para no generar conflictos ya que estos scripts permiten facilitar el uso del sistema pero no lo automatiza.

Al igual que el script anterior se divide entres partes únicamente cambiando en la primera y segunda parte la ruta donde se almacenan y se crean los diferentes directorios, el cambio más importante lo vemos en la tercera parte en donde a diferencia de únicamente realizar la solicitud de la firma, usando el fichero propio de configuración de la CA_intermediate, realizaremos la firma con el uso de la función *ca* para la cual necesitaremos aportar un archivo de configuración como los que hemos analizado anteriormente en este caso usaremos el fichero de configuración *openssl.cnf* . También aportamos las distintas configuraciones que deseamos para el certificado que estamos generando como el número de días por los que será válido (opción *-day*) y la extensión para este certificado (opción *-extensions*) en este caso *v3_intermediate_ca*, creada en el fichero *openssl.cnf* como se ve en la siguiente figura.

```
[ v3_intermediate_ca ]


# Extensions for a typical CA


# PKIX recommendation.

subjectKeyIdentifier=hash

authorityKeyIdentifier=keyid:always,issuer

basicConstraints = critical,CA:true

# Key usage: this is typical for a CA certificate. However since it will
# prevent it being used as an test self-signed certificate it is best
# left out by default.
keyUsage = critical, digitalSignature, cRLSign, keyCertSign

# Some might want this also
# nsCertType = sslCA, emailCA

# Include email address in subject alt name: another PKIX recommendation
# subjectAltName=email:copy
# Copy issuer details
# issuerAltName=issuer:copy

# DER hex encoding of an extension: beware experts only!
# obj=DER:02:03
# Where 'obj' is a standard or added object
# You can even override a supported extension:
# basicConstraints= critical, DER:30:03:01:01:FF
```

Figura 17: Extensión v3_intermediate_ca en el fichero openssl.cnf

El último paso tras crear y obtener el certificado será la creación de la cadena de certificados (*chain.cert.pem*) que permitirá la comprobación de los certificados emitidos por las CA_intermediate.

Como hemos comentado antes crearemos tres CA_intermediate, por lo que ejecutaremos el script tres veces diferentes con los siguientes valores.

1. **Comunicaciones** **1001**
2. **Seguridad** **2001**
3. **Privacidad** **3001**

Tras la ejecución del script las tres veces obtenemos la siguiente estructura.

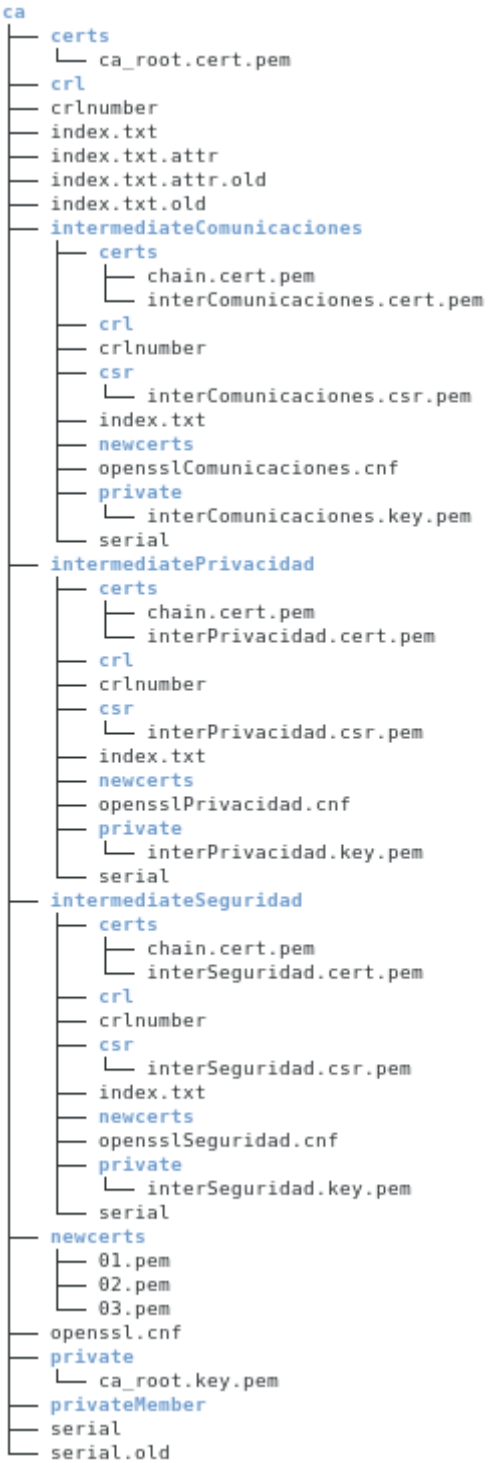


Figura 18: Estructura de Certificados tras ejecuitar CA_intermediate.sh

El último paso será la creación de los certificados de los miembros de la organización para lo que haremos uso de dos scripts diferentes uno será el encargado de crear la clave privada del miembro haciendo uso de la función *genrsa* y el segundo script será el que genera los certificados para las distintas comunidades.

Comencemos por el script de generación de claves privadas este es denominado *createKey.sh* y podemos observarlo en la siguiente figura.

```
OWNER_PATH=$1 #identificador del miembro

cd ca

openssl genrsa -out privateMember/$OWNER_PATH.key.pem 2048
chmod 400 privateMember/$OWNER_PATH.key.pem
```

Figura 19: Script createKey.sh

Podemos ver que este script necesita el paso de un atributo que es el identificador del miembro para el cual vamos a generar la clave privada. Al igual que ocurría con las claves generadas para las autoridades certificadoras una vez generada la clave se alteran los permisos de esta para solo permitir la lectura puesto que no se necesita realizar ninguna otra acción sobre ella.

El segundo script es de generación de certificados denominado *createCerts.sh* y funciona de similar manera al script encargado de generar las autoridades certificadoras intermedias, podemos verlo a continuación.

```
OWNER_PATH=$1 #identificador de miembro
NAME_PATH=$2 #nombre completo del miembro
COM_PATH=$3 #comunidad para la cual es el certificado

#NO TOUCH
dir_interCA=intermediate$COM_PATH
config_PATH=intermediate$COM_PATH/openssl$COM_PATH.cnf
DATA_PATH="/C=ES/ST=CastillaLeon/L=Valladolid/O=DigIdCom/OU=$COM_PATH/CN=$NAME_PATH"

cd ca

openssl req -config $config_PATH -new -sha256 -key privateMember/$OWNER_PATH.key.pem -subj $DATA_PATH -out $dir_
interCA/csr/$OWNER_PATH.csr.pem

openssl ca -config $config_PATH -batch -days 1000 -notext -md sha256 -in $dir_interCA/csr/$OWNER_PATH.csr.pem -o
ut $dir_interCA/certs/$OWNER_PATH.cert.pem

chmod 444 $dir_interCA/certs/$OWNER_PATH.cert.pem
```

Figura 20: Script createCerts.sh

Este último script necesita del paso de tres atributos que tal y como se describen en la imagen corresponden al identificador del miembro, el nombre completo del mismo y la comunidad para la que se desea emitir el certificado. El primer paso que realiza el script es la generación de la solicitud de firma haciendo uso del archivo de configuración de la CA_intermediate correspondiente a la comunidad para la cual desea el certificado y seguidamente pasa a ser firmado por la CA_intermediate haciendo uso del mismo fichero de configuración comentado anteriormente. Para terminar una vez ha sido creado el certificado a éste se le alteran los permisos para permitir solo la lectura.

Siguiendo lo que hemos explicado al inicio de este apartado ahora generamos las claves y certificados de los tres miembros y lo haremos teniendo en cuenta la pertenencia a las comunidades por parte de los miembros de la siguiente manera:

- **Alice** pertenece a las comunidades de **Comunicaciones** y **Privacidad**.
- **Bob** pertenece a las comunidades de **Seguridad** y **Privacidad**.
- **Mallet** pertenece a las comunidades de **Comunicaciones** y **Seguridad**.

Ejecutamos el primer script tres veces, una por cada miembro, y seis veces el segundo, dos por cada miembro. El resultado que obtenemos es el siguiente:

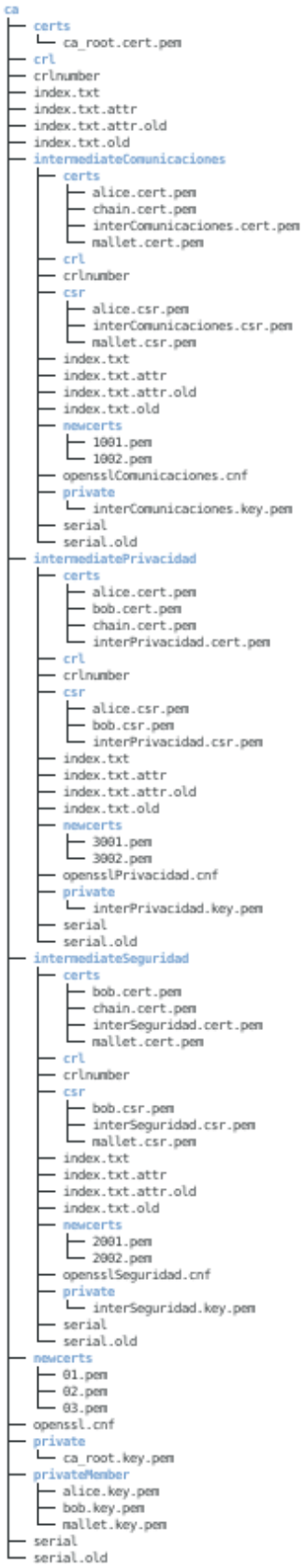


Figura 21: Estructura de Certificados completa

Lo último que nos faltaría hacer es comprobar la validez de cada uno de los certificados para asegurarnos que nuestro sistema es funcional.

Para ello haremos uso de la herramienta *verify* y usaremos las cadenas de certificados que hemos generada al crear nuestras CA_intermediate, aunque posean el mismo nombre cada CA_intermediate tiene la suya propia y diferente, esto es así para simplificar el uso de la herramienta *verify* ya que es la única funcionalidad de las cadenas de certificados.

```
root@DigIdCom:/DigIdCom# openssl verify -CAfile ca/intermediateComunicaciones/certs/chain.cert.
pem ca/intermediateComunicaciones/certs/alice.cert.pem
ca/intermediateComunicaciones/certs/alice.cert.pem: OK
root@DigIdCom:/DigIdCom# openssl verify -CAfile ca/intermediatePrivacidad/certs/chain.cert.pem
ca/intermediatePrivacidad/certs/alice.cert.pem
ca/intermediatePrivacidad/certs/alice.cert.pem: OK
```

Figura 22: Verificación certificados de Alice

```
root@DigIdCom:/DigIdCom# openssl verify -CAfile ca/intermediateSeguridad/certs/chain.cert.pem
ca/intermediateSeguridad/certs/bob.cert.pem
ca/intermediateSeguridad/certs/bob.cert.pem: OK
root@DigIdCom:/DigIdCom# openssl verify -CAfile ca/intermediatePrivacidad/certs/chain.cert.pem
 ca/intermediatePrivacidad/certs/bob.cert.pem
ca/intermediatePrivacidad/certs/bob.cert.pem: OK
```

Figura 23: Verificación certificados de Bob

```
root@DigIdCom:/DigIdCom# openssl verify -CAfile ca/intermediateComunicaciones/certs/chain.cert.pem
ca/intermediateComunicaciones/certs/mallet.cert.pem
ca/intermediateComunicaciones/certs/mallet.cert.pem: OK
root@DigIdCom:/DigIdCom# openssl verify -CAfile ca/intermediateSeguridad/certs/chain.cert.pem ca/in
termediateSeguridad/certs/mallet.cert.pem
ca/intermediateSeguridad/certs/mallet.cert.pem: OK
```

Figura 24: Verificación certificados de Mallet

Como podemos ver en las figuras anteriores todos los miembros disponen de dos certificados en activo necesitando únicamente hacer uso de una única clave privada por cada miembro en lugar de por cada certificado. Por lo que podemos afirmar que la Estructura de Certificados logra completar la que es su función.

3. Base de Datos

Ahora nos encontramos frente a la segunda mitad de nuestro sistema, y será la parte encargada de describir cual es el estado actual de la organización y almacenar todos los datos relacionado con esto.

Como hemos hecho durante la explicación del Esquema de Certificados, empezaremos comentando cuál ha sido la tecnología empleada para la implementación de esta parte y luego procederemos a explicar cómo ha sido esta implementación.

3.1 Tecnología empleada

Para la implementación de esta última parte de nuestro sistema hemos decidido emplear el sistema gestor de bases de datos MySQL en su versión versión 8.0.29.

La decisión de apostar por emplear MySQL y no otro gestor podemos resumirla de la misma manera que la decisión de emplear OpenSSL para el Esquema de Certificados. Al emplear MySQL podemos reducir los costes de nuestro proyecto ya que la versión que hemos empleado es gratuita y cuenta con las características suficientes para cumplir con las exigencias de nuestro sistema, otra de las razones para emplear MySQL es que es compatible con los sistemas operativos más usados a día de hoy como son Windows, Linux y MacOs, por último y no menos importante es la opción que tiene MySQL es la opción de ser empleado haciendo uso de comandos de terminal cargando a MySQL archivos de tipo SQL.

Para la visualización del contenido de las tablas hemos hecho uso del Workbench del propio MySQL, no es necesario emplearlo ya que podremos visualizar nuestra base de datos directamente desde terminal, pero tratándose de una herramienta gratuita lo emplearemos para facilitar la tarea de visualizar el contenido de las diversas tablas que componen nuestra base de datos, si una organización quisiera implantar nuestro sistema y los recursos de sus máquinas fueran más limitados podría emplear el sistema sin la necesidad de esta herramienta.La elección de emplear la versión 10 de Debian en lugar de la versión 11, la cual es la más actual, reside principalmente en las incompatibilidades que esta versión tiene con alguna de las herramientas que consideramos la más adecuadas para la implementación de nuestro sistema. También

3.2 Configuraciones aplicadas

A diferencia de como ocurría en OpenSSL no han sido necesarias excesivas configuraciones para poder hacer uso de MySQL y su Workbench para emplearlo dentro de nuestro sistema.

Las configuraciones que hemos necesitado emplear no han sido otras que las necesarias para instalar MySQL en nuestra máquina y las necesarias para vincular correctamente el Workbench con MySQL. LA configuración principal que hemos realizado a la hora de instalar MySQL fue definir un método de autentificación para el usuario *root* del gestor de base de datos, ya que recién realizamos la instalación obtendremos un error que dice:

Failed! Error: SET PASSWORD has no significance for user 'root'@'localhost' as the authentication method used doesn't store authentication data

Este error nos informa de que lo que comentábamos anteriormente, no existe ningún método para autentificar al usuario *root*, para resolverlo solo tenemos que seguir dos pasos muy sencillos desde terminal que son:

1. *sudo mysql*

2. Estando dentro de la consola de MySQL:

*ALTER USER 'root'@'localhost' IDENTIFIED WITH mysql_native_password by '****';*

Con esto tendríamos listo MySQL, el siguiente paso fue vincularlo con el Workbench, lo cual se hace todo de manera automática salvo un paso puesto que es necesario modificar los permisos de la aplicación debemos habilitar las opciones que permitan a la aplicación leer, cambiar, añadir y eliminar contraseñas guardadas.

Los permisos quedarían de la siguiente forma.

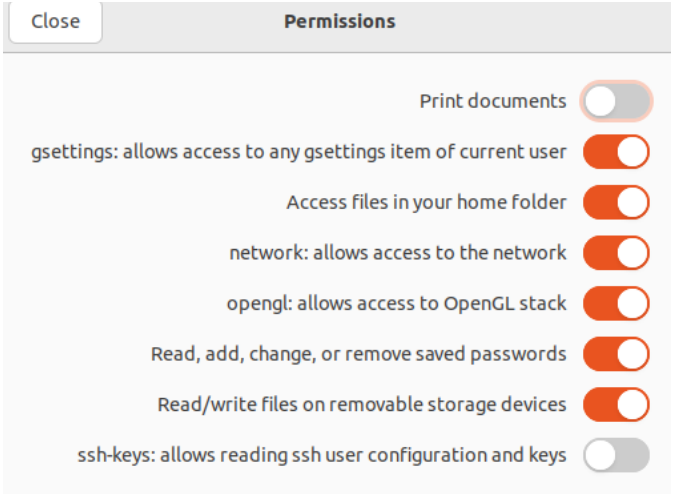


Figura 25: Permisos de MySQL WorkBench

3.3 Construcción

Para la construcción de nuestra base de datos deberemos de crear dentro de MySQL la base de datos que emplearemos, nosotros la nombramos como nuestro sistema DigIdCom, y dentro de la base de datos crearemos todas las tablas que han surgido durante nuestra fase de diseño y definiremos tanto las claves primarias como claves foráneas que identificamos en dicha fase. Todo esta construcción de tablas está recogida en el fichero *creatorDB.sql* para el cual también hemos desarrollado un pequeño script para facilitar la ejecución de todas las sentencias que se recogen en el archivo SQL y así poder crear nuestra base de datos. Comenzaremos por explicar todas las sentencias del archivo anteriormente mencionado.

En primer lugar debemos preparar nuestro gestor para recibir nuestras sentencias por lo que desactivamos de manera temporal la detección de claves foráneas y creamos la base de datos DigIdCom y seguidamente seleccionamos la base de datos que vamos a emplear la cual no es otra que DigIdCom.

```
2 SET foreign_key_checks = 0; # eliminamos la detección de dependencias
3
4 create database DigIdCom;
5
6 USE DigIdCom;
```

Figura 26: Preparación del gestor para recibir las sentencias SQL

Los siguientes pasos consisten en crear todas y cada uno de las tablas, lo cual lo haremos siguiendo el siguiente patrón:

1. Eliminamos la tabla en caso de que existiera.
2. Creamos la tabla.
3. Definimos la clave primaria (primary key).
4. Definimos las claves foráneas (foreign key) en caso de tenerlas.

Ahora mostraremos las sentencias para la creación de cada tabla.

MIEMBRO

```
 8 drop table if exists miembro;
 9 create table miembro(
10         nombreApellidos varchar(50) not null, ID varchar(10),
11         constraint mie_cp primary key (ID)
12 );
```

Figura 27: Creación de la tabla miembro.

GESTOR

```
14 drop table if exists gestor;
15 create table gestor(
16         nombreApellidos varchar(50) not null, ID varchar(10),
17         constraint ges_cp primary key (ID)
18 );
```

Figura 28: Creación de la tabla gestor

COMUNIDAD

```
20 drop table if exists comunidad;
21 create table comunidad(
22         nombreComunidad varchar(20),fechaCreacion varchar(70),
23         creador varchar(10), numMiembros integer check (numMiembros >= 0),
24         constraint com_cp primary key (nombreComunidad),
25         constraint com_cf foreign key (creador) references gestor(ID)
26
27 );
```

Figura 29: Creación de la tabla comunidad

CA_ROOT

```
29 drop table if exists ca_root;
30 create table ca_root(
31         nombreOrg_cert varchar(10) not null, ubicacion varchar(30) not null,
32         serialNumber_root varchar(70),
33         valida_root varchar(1) check (valida_root in ('s', 'n')),
34         fechaFin_root varchar(25),
35         constraint car_cp primary key (serialNumber_root)
36 );
```

Figura 30: Creación de la tabla ca_root

CA_INT

```
38 drop table if exists ca_int;
39 create table ca_int(
40         nombreCom_int varchar(20), ubicacion_Com varchar(30),
41         serialNumber_int varchar(70),
42         valida varchar(1) check (valida in ('s', 'n')),
43         fechaFin_int varchar(25),
44         constraint cai_cp primary key (serialNumber_int),
45         constraint cai_cf foreign key (nombreCom_int) references comunidad(nombreComunidad)
46 );
```

Figura 31: Creación de la tabla ca_int

CERTIFICADO

```
48 drop table if exists certificado;
49 create table certificado(
50         serialNumber varchar(70),
51         revocado varchar(1) check (revocado in ('s', 'n')),
52         fechaFin varchar(25),
53         constraint cer_cp primary key (serialNumber)
54 );
```

Figura 32: Creación de la tabla certificado

es_propietario_de

```
56 drop table if exists es_propietario_de;
57 create table es_propietario_de(
58         ID varchar(10), serialNumber varchar(70),
59         constraint esp_cp primary key (serialNumber),
60         constraint esp_cf foreign key (ID) references miembro(ID),
61         constraint esp_cf2 foreign key (serialNumber) references certificado(serialNumber)
62 );
```

Figura 33: Creación de la tabla es_propietario_de

certifica

```
64 drop table if exists certifica;
65 create table certifica(
66         serialNumber_int varchar(70), nombreComunidad varchar(20),
67         constraint certi_cp primary key (serialNumber_int),
68         constraint certi_cf foreign key (nombreComunidad) references comunidad(nombreComunidad),
69         constraint certi_cf2 foreign key (serialNumber_int) references ca_int(serialNumber_int)
70 );
```

Figura 34: Creación de la tabla certifica

emite

```
72 drop table if exists emite;
73 create table emite(
74         serialNumber_int varchar(70), serialNumber varchar(70),
75         constraint emi_cp primary key (serialNumber),
76         constraint emi_cf foreign key (serialNumber_int) references ca_int(serialNumber_int),
77         constraint emi_cf2 foreign key (serialNumber) references certificado(serialNumber)
78 );
```

Figura 35: Creación de la tabla emite

crea

```
80 drop table if exists crea;
81 create table crea(
82         serialNumber_root varchar(70), serialNumber_int varchar(70),
83         constraint cre_cp primary key (serialNumber_int),
84         constraint cre_cf foreign key (serialNumber_int) references ca_int(serialNumber_int),
85         constraint cre_cf2 foreign key (serialNumber_root) references ca_root(serialNumber_root)
86 );
```

Figura 36: Creación de la tabla crea

Con esto tendríamos todas las sentencias necesarias para levantar nuestra base de datos, para ello usamos el siguiente script que únicamente nos necesita que le pasemos el usuario de MySQL que ejecuta las inserciones, el script es el nombrado como *creatorDB.sh* y tiene la siguiente forma.

```
EXE_PATH=$1

mysql -u $EXE_PATH -p < creatorDB.sql
```

Figura 37: Contenido de creatorDB.sh

Al ejecutar este script el gestor MySQL solicitará la contraseña del usuario que accede a la base de datos, el cual es el que le aportamos en la variable EXE_PATH.

Tras ejecutar el script tendremos lista la base de datos de nuestro sistema y si observamos el contenido de la base de datos DigIdCom en el Workbench de MySQL veremos que todas las tablas han sido creadas correctamente.

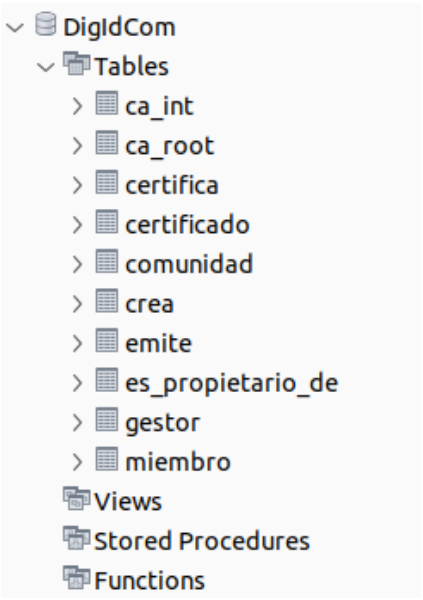


Figura 38: Resultado de la ejecución del script creatorDB.sh

Con ambas partes listas ahora podemos observar en la siguiente sección cómo ambas se unen para que trabajen de manera conjunta y mantener el contenido de ambas partes coherente.

4. Integración de las partes

Ahora mostraremos cómo trabajan ambas partes en conjunto y cómo actúan estos para ello tomaremos una versión más reducida del ejemplo de la **sección 2** de este mismo capítulo. El ejemplo que emplearemos en esta ocasión consta de una organización denominada DigIdCom con dos comunidades, las cuales son Comunicaciones y Seguridad respectivamente, y dos miembros conocidos como Alice Smith y Bob Trenton, de los cuales Alice actuará también como gestor.

4.1 Funciones principales

Obviamos la creación de la base de datos puesto que no existe ninguna variación con lo que hemos mostrado en la **sección 3**. Por lo que comenzaremos por crear la CA raíz.

- **Creación de la CA raíz**

1. sh CA_Root.sh <ID del ABD>
    - *sh CA_Root.sh root*

Al igual que ocurre con el script *creatorDB.sh* que mostrabamos anteriormente este tambien es un script de puesta en marcha del sistema por lo que será necesaria la colaboración del ABD (Administrador de Base de Datos).

Este script será el encargado de generar toda la estructura de directorios de la CA raíz así como el certificado digital autofirmado, que necesita para operar, y los ficheros que se encargarán de recoger cuales son los certificados emitidos por esta CA y el estado en se encuentran los certificados emitidos por ella. Lo segundo que hará este script será poblar la tabla ca_root de la base de datos de manera que tras la ejecución de este script obtenemos lo siguiente.

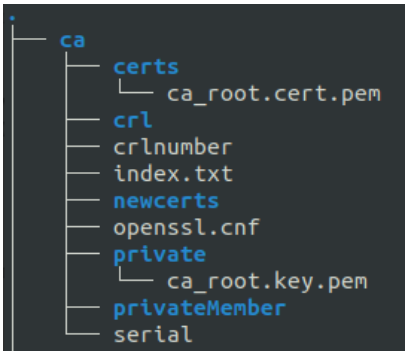


Figura 39: Estado de la Estructura de Certificados tras CA_root.sh

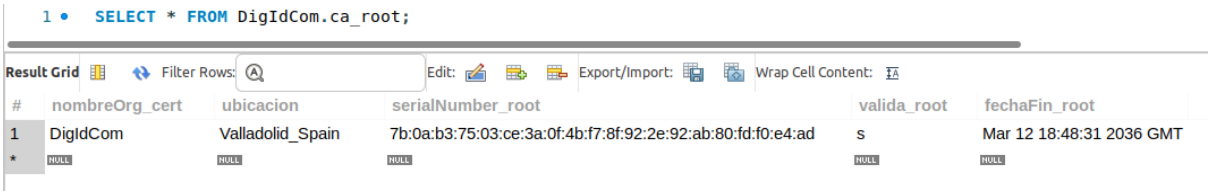


Figura 40: Estado de la BD (Base de Datos) tras CA_root.sh

- **Creación del primer gestor**

1. sh createMember.sh <Nombre_Apellido> <ID> <Contraseña> <rol> <gestor encargado>

    - *sh createMember.sh Alice_Smith alice 0000 gestor root*

Con este script lograremos generar el registro en la BD del miembro en cuestión pero al tratarse de un gestor también queda registrado como y se habrá generado un usuario para la BD tal con los permisos especificados en la matriz de accesos y los permisos necesarios para crear otros miembros y gestores.

También en la Estructura de Certificados veremos que se ha generado su clave privada.

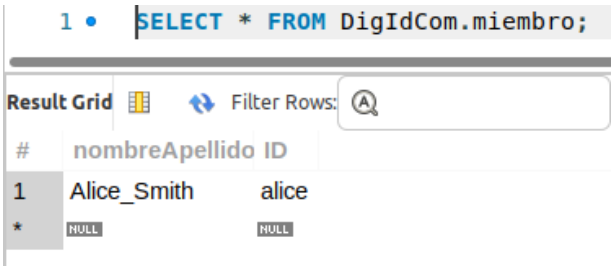


Figura 41: Estado de tabla miembro de la BD tras createMember.sh

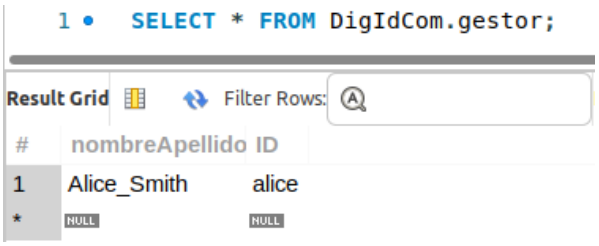


Figura 42: Estado de tabla gestor de la BD tras createMember.sh

Con esto tendríamos lista la base de nuestro sistema y ya podremos manipular nuestro sistema sin la necesidad de recurrir a nuestro ABD.

- **Creación de comunidades**

1. sh CA_intermediate.sh <Nombre de la comunidad> <Serial> <gestor encargado>

    - *sh CA_intermediate.sh Comunicaciones 1000 alice*
    - *sh CA_intermediate.sh Seguridad 2000 alice*

Como hemos dicho creamos dos comunidades para este ejemplo por lo que empleamos el script dos veces de manera consecutiva haciendo que se genere la estructura de directorios dentro de la CA raíz referente a cada una CA intermedias que darán soporte a cada comunidad vemos también que hemos definido el serial que determina cual es el primer número de serie para los certificados que emitió dicha CA intermedia. A la hora de determinar este serial debemos ser previsores ya que si una comunidad comienza con el número de serie 1000 y la siguiente comunidad lo hace con el 2000 esto dejará disponibilidad de mil certificados para la primera comunidad, como hemos explicado anteriormente estos scripts no automatizan el uso del sistema sino que lo facilitan por lo que debemos ser conscientes de las acciones que realizamos.

Al ejecutar estos scripts también alteramos la BD de manera que queda registrada la nueva comunidad, la nueva CA intermedia y quien ha sido la CA raíz que ha generado esa CA intermedia.

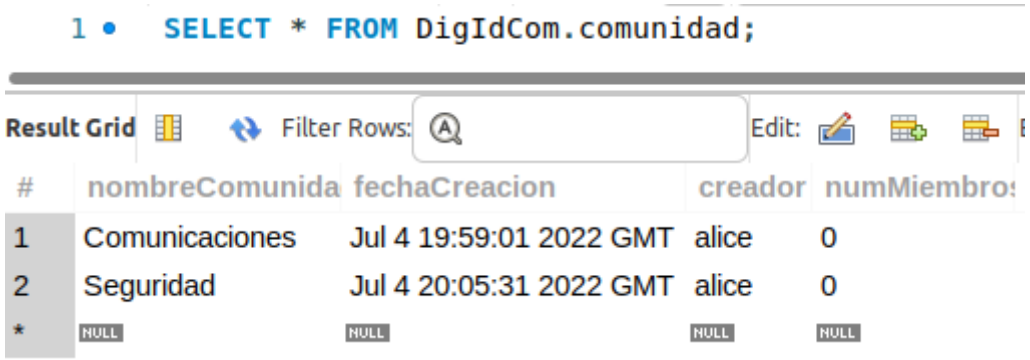


Figura 43: Estado de la tabla comunidad en la BD tras la ejecución de CA_intermediate.sh

```
1 •  SELECT * FROM DigIdCom.ca_int;
```

Result Grid | Filter Rows: | Edit: | Export/Import:

#	nombreCom_int	ubicacion_Com	serialNumber_in	valida	fechaFin_int
1	Comunicaciones	Valladolid_Spain	1 (0x1)	s	Mar 12 19:59:01 2036 GMT
2	Seguridad	Valladolid_Spain	2 (0x2)	s	Mar 12 20:05:31 2036 GMT

Figura 44: Estado de la tabla ca_int en la BD tras la ejecución de CA_intermediate.sh

```
1 •  SELECT * FROM DigIdCom.certifica;
```

Result Grid | Filter Rows:

#	serialNumber_in	nombreComunida
1	1 (0x1)	Comunicaciones
2	2 (0x2)	Seguridad

Figura 45: Estado de la tabla certifica en la BD tras la ejecución de CA_intermediate.sh

```
1 •  SELECT * FROM DigIdCom.crea;
```

Result Grid | Filter Rows: | Edit: | Export/Im

#	serialNumber_root	serialNumber_int
1	7b:0a:b3:75:03:ce:3a:0f:4b:f7:8f:92:2e:92:ab:80:fd:f0:e4:ad	1 (0x1)
2	7b:0a:b3:75:03:ce:3a:0f:4b:f7:8f:92:2e:92:ab:80:fd:f0:e4:ad	2 (0x2)

Figura 46: Estado de la tabla crea en la BD tras la ejecución de CA_intermediate.sh

- **Creación de un nuevo miembro**

1. sh createMember.sh <Nombre_Apellido> <ID> <Contraseña> <rol> <gestor encargado>

    - *sh createMember.sh Bob_Trenton 0000 miembro alice*

El funcionamiento de este script ya lo hemos podido ver a la hora de crear a nuestra gestora Alice la única diferencia que veremos es que ahora solo veremos añadido a Bob en la tabla de miembro y no en la de gestor puesto que no es el rol que le corresponde.

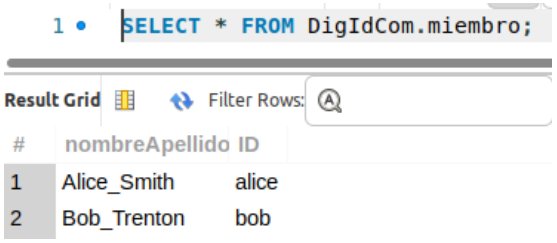


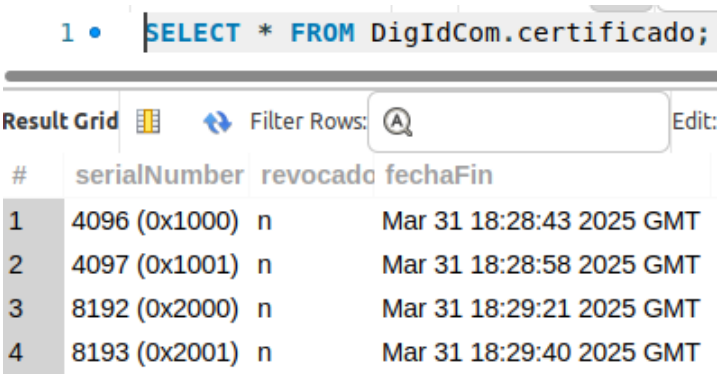
Figura 47: Estado de la tabla miembro de la BD tras createMember.sh

- **Creación de certificados**

1. sh createCerts.sh <ID del propietario> <Comunidad> <gestor encargado>

    - *sh createCert.sh alice Comunicaciones alice*
    - *sh createCert.sh alice Seguridad alice*
    - *sh createCert.sh bob Comunicaciones alice*
    - *sh createCert.sh bob Seguridad alice*

Cuando ejecutamos este comando vemos como el miembro en cuestión recibe el certificado para la comunidad designada, sino que también son pobladas todas las tablas de la BD que se encargan de recoger toda la información acerca del certificado, quien lo ha emitido y quien es el dueño de este.

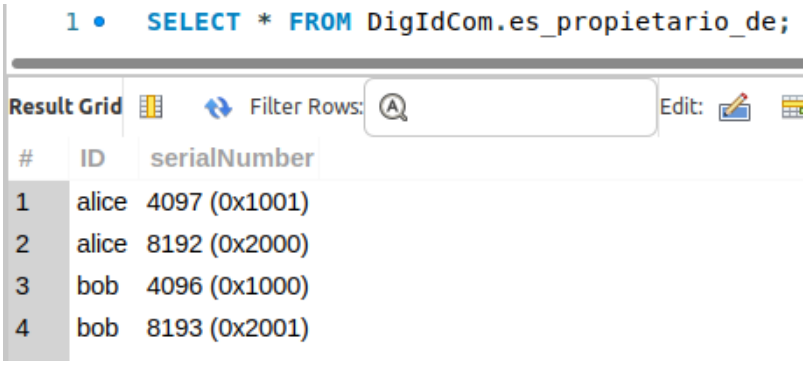


```
1 •  SELECT * FROM DigIdCom.certificado;
```

Result Grid | Filter Rows: | Edit:

#	serialNumber	revocado	fechaFin
1	4096 (0x1000)	n	Mar 31 18:28:43 2025 GMT
2	4097 (0x1001)	n	Mar 31 18:28:58 2025 GMT
3	8192 (0x2000)	n	Mar 31 18:29:21 2025 GMT
4	8193 (0x2001)	n	Mar 31 18:29:40 2025 GMT

Figura 48: Estado de la tabla certificado de la BD tras createCerts.sh

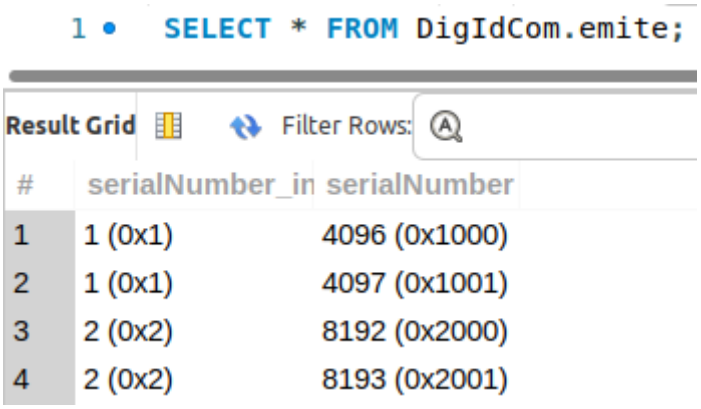


```
1 •  SELECT * FROM DigIdCom.es_propietario_de;
```

Result Grid | Filter Rows: | Edit:

#	ID	serialNumber
1	alice	4097 (0x1001)
2	alice	8192 (0x2000)
3	bob	4096 (0x1000)
4	bob	8193 (0x2001)

Figura 49: Estado de la tabla es_propietario_de de la BD tras createCerts.sh



```
1 •  SELECT * FROM DigIdCom.emite;
```

Result Grid | Filter Rows:

#	serialNumber_in	serialNumber
1	1 (0x1)	4096 (0x1000)
2	1 (0x1)	4097 (0x1001)
3	2 (0x2)	8192 (0x2000)
4	2 (0x2)	8193 (0x2001)

Figura 50: Estado de la tabla emite de la BD tras createCerts.sh

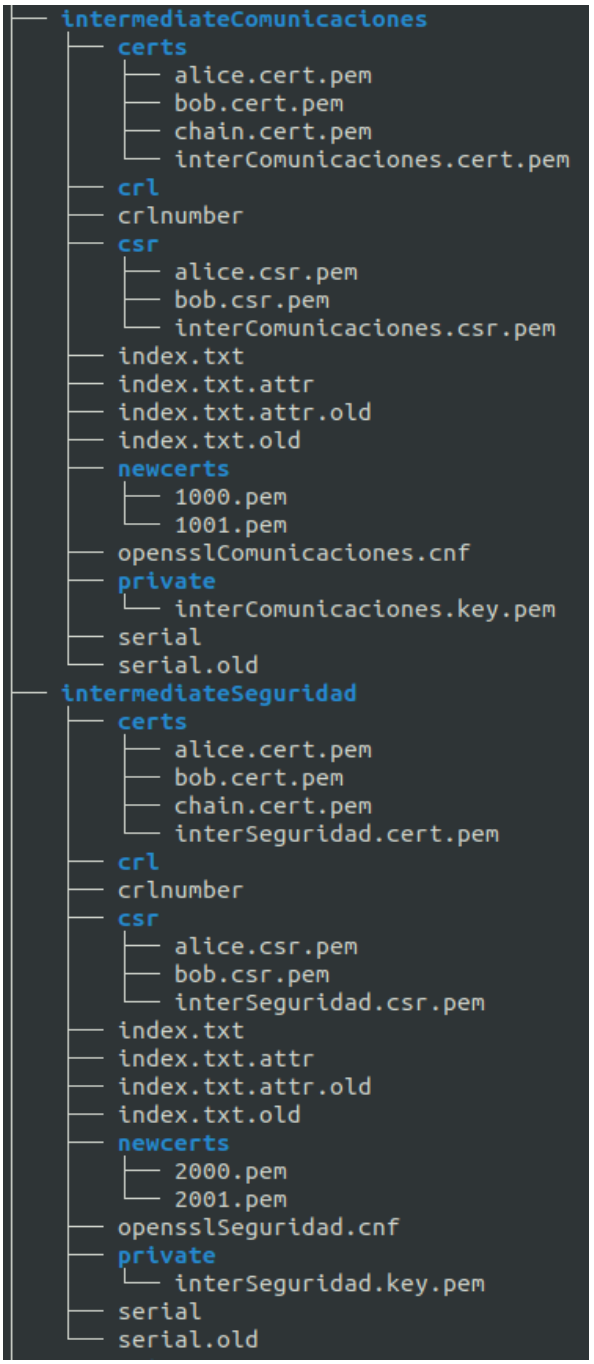


Figura 51: Estado del Esquema de Certificados tras createCerts.sh

- **Verificar un certificado**

1. sh verifyCert.sh <ID del miembro> <Comunidad>

    - *sh verifyCert.sh alice Comunicaciones*
    - *sh verifyCert.sh alice Seguridad*

Con la ayuda de este script podremos verificar cada uno de los certificados de un miembro dentro de la organización, únicamente deberemos indicar el miembro y la comunidad para comprobarlo. Esto lo puede realizar haciendo uso internamente de la herramienta verify de OpenSSL.

Tras ejecutar ambos comando podemos ver que obtenemos lo siguiente.

```
root@aaron-Strix:/DigIdCom# sh verifyCert.sh alice Comunicaciones
/DigIdCom/ca/intermediateComunicaciones/certs/alice.cert.pem: OK
root@aaron-Strix:/DigIdCom# sh verifyCert.sh alice Seguridad
/DigIdCom/ca/intermediateSeguridad/certs/alice.cert.pem: OK
```

Figura 52: Resultado del script verifyCert.sh

Hasta este punto hemos logrado poner en funcionamiento el sistema de manera muy básica, registramos nuevos miembros y gestores, creamos nuevas comunidades, añadimos nuevos miembros a ellas generando los respectivos certificados, …

Pero como todos sabemos la vida dentro de cualquier organización está en constante cambio y evolución por lo que ahora vamos a mostrar y explicar cómo se comporta el sistema frente a esta evolución y cambio; para mostrarlo planteamos una serie de escenarios consecutivos.
El tiempo en nuestra organización transcurre con normalidad, pero se observa que los progresos del miembro Bob Trenton en la comunidad de Seguridad son escasos, por lo que se toma la decisión de darlo de baja de esta comunidad para que enfoque su esfuerzo en la otra comunidad de la que este es miembro. Para Bob el haber sido retirado del proyecto de seguridad resulta un golpe duro por lo que consecuencia de esto es que su rendimiento general disminuye de manera notable, por lo que la dirección determina que lo mejor es la expulsión de este miembro de la organización. Los problemas se suceden en nuestra organización y la comunidad de Seguridad deja de ser viable por lo que se determina su clausura.

- **Dar de baja un miembro de una comunidad**

1. sh revokeCertsUser.sh <ID del miembro> <Comunidad / todos> <gestor encargado>
2. sh dropCert.sh <ID del miembro> <Comunidad / todos> <gestor encargado>

    - *sh revokeCertsUser.sh bob Seguridad alice*
    - *sh dropCert.sh bob Seguridad alice*

Para eliminar el miembro de la comunidad, en este caso Bob y de la comunidad Seguridad, deberemos de realizar dos pasos en primer revocamos el certificado que pueda tener activo y actualizamos su estado en la BD, en caso de no tener ninguno no habrá ningún problema, y seguidamente eliminamos de la BD todos los certificados del miembro asociados a la comunidad en cuestión. Al no tener ya en la BD, que recordemos que es la encargada de definir el estado actual de la organización, ningún certificado que lo relacione con esa comunidad podemos afirmar que ese miembro ya no pertenece a esa comunidad. En el Esquema de Certificados veríamos generado el archivo crl que nos indica que ese certificado no es válido y el fichero index.txt correspondiente a esa CA intermedia también se habrá actualizado.

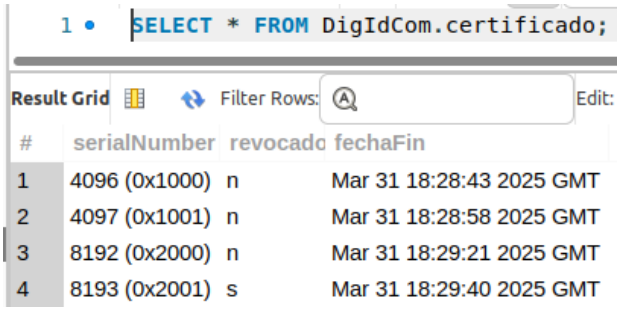


Figura 53: Estado de la tabla certificado de la BD tras revokeCertsUser.sh

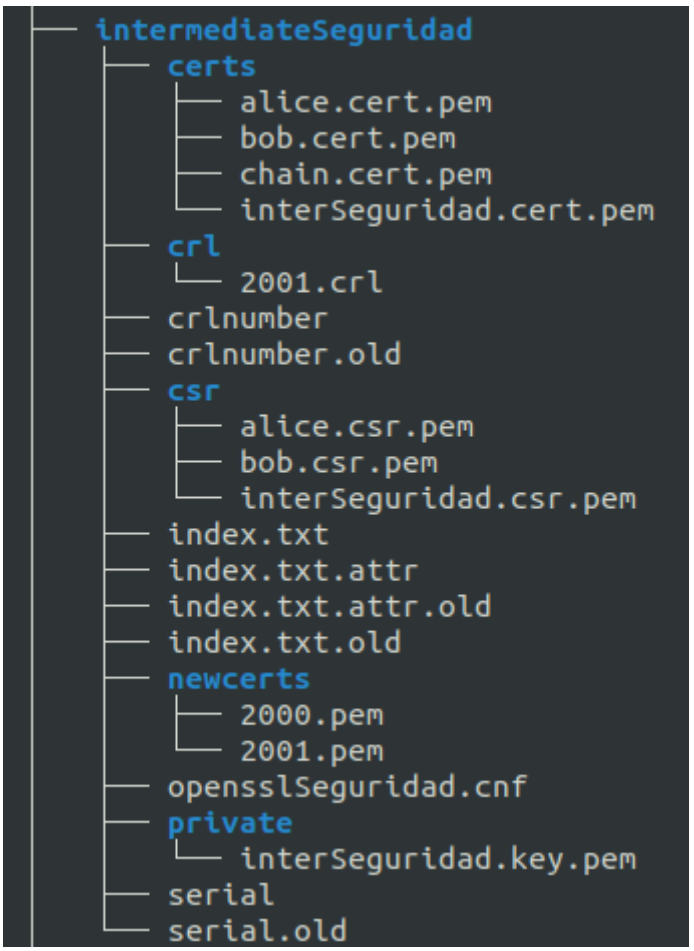


Figura 54: Estado del Esquema de Certificados tras revokeCertsUser.sh

Tras esto sigue la eliminación y podemos observar que no existe rastro del certificado de Bob para seguridad en la BD.

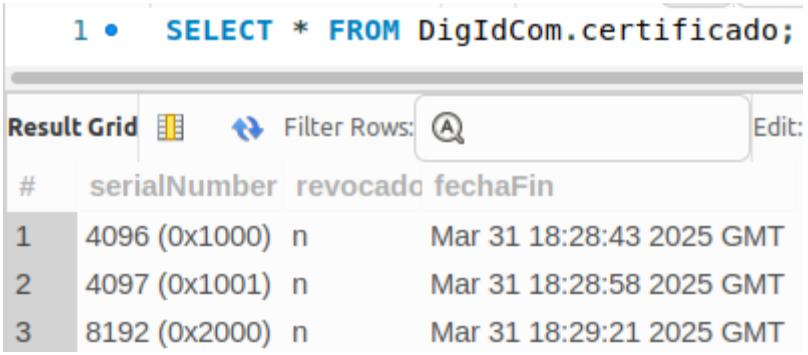


Figura 55: Estado de la tabla certificado de la BD tras dropCert.sh

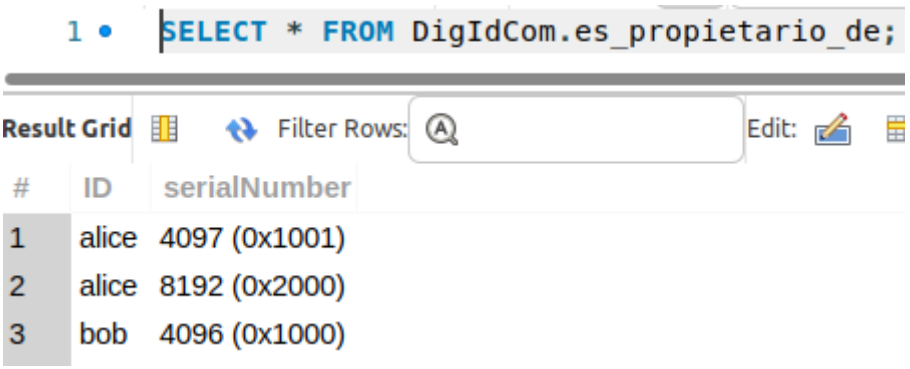


Figura 56: Estado de la tabla es_propietario_de de la BD tras dropCert.sh

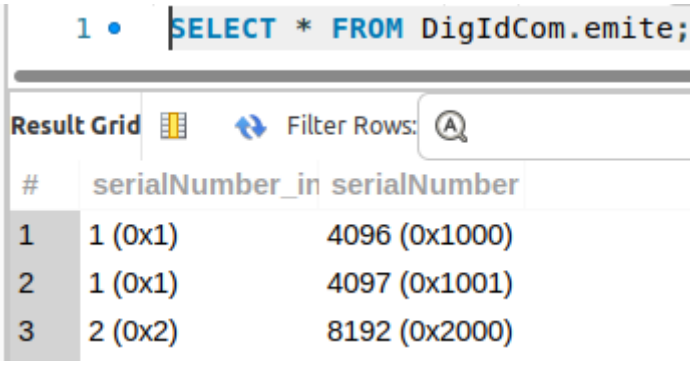


Figura 57: Estado de la tabla emite de la BD tras dropCert.sh

- **Eliminar un miembro de la organización**

1. sh revokeCertsUser.sh <ID del miembro> <Comunidad / todos> <gestor encargado>
2. sh dropCert.sh <ID del miembro> <Comunidad / todos> <gestor encargado>
3. sh deleteMember.sh <ID del miembro> <gestor encargado>

    - sh revokeCertsUser.sh bob all alice
    - sh dropCert.sh bob all alice
    - sh deleteMember.sh bob alice

Eliminar a un miembro de la organización es equivalente a darlo de baja de todas las comunidades a las que este pertenezca para seguidamente eliminar el miembro de manera literal de la BD y de los usuarios de la BD. Naturalmente seguiremos con el mismo escenario que hemos visto anteriormente, por lo que tras ejecutar todos los comandos obtenemos lo siguiente.

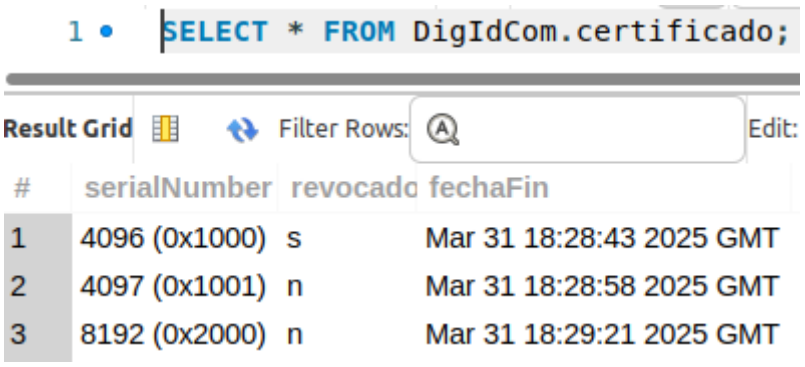


Figura 58: Estado de la tabla certificado de la BD tras revokeCertsUser.sh

Ahora como sucedía anteriormente seguirá la eliminación

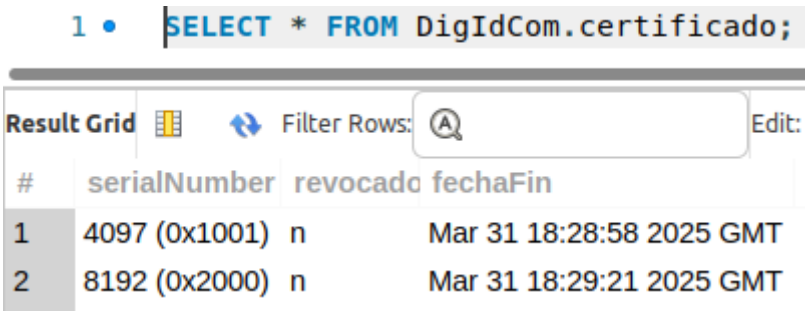


Figura 59: Estado de la tabla certificado de la BD tras dropCert.sh

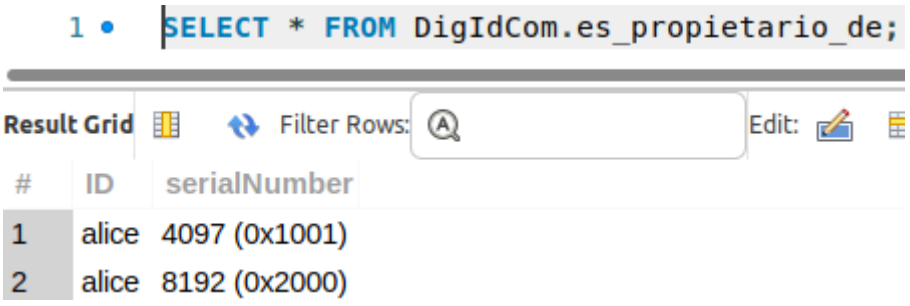


Figura 60: Estado de la tabla es_propietario_de de la BD tras dropCert.sh

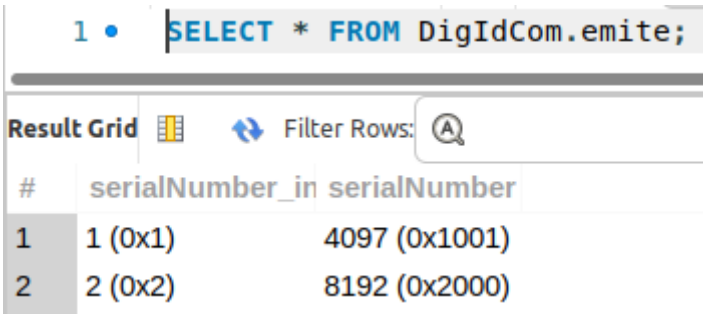


Figura 61: Estado de la tabla emite de la BD tras dropCert.sh

Y ahora eliminamos de manera definitiva el miembro, como venimos haciendo solo lo eliminamos de la descripción actual del sistema, en otras palabras se eliminará de la BD y no de la Estructura de Certificados.

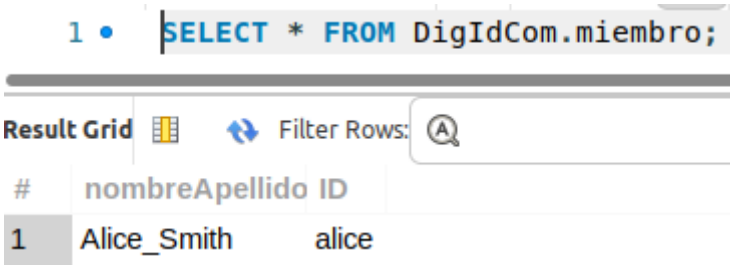


Figura 62: Estado de la tabla miembro de la BD tras deleteMember.sh

- **Eliminar una comunidad de la organización**

1. sh revokeCertsCA. sh <Comunidad> <gestor encargado>
2. sh dropCertCA.sh <Comunidad> <gestor encargado>

    - sh revokeCerts.sh Seguridad alice
    - sh dropCertsCA.sh Seguridad alice

Para eliminar una comunidad, en este ejemplo Seguridad, el proceso como se puede ver es muy similar a eliminar un miembro de la organización primero revocamos los certificados emitidos por la CA intermedia asociada a esa comunidad y el propio certificado de dicha CA intermedia.

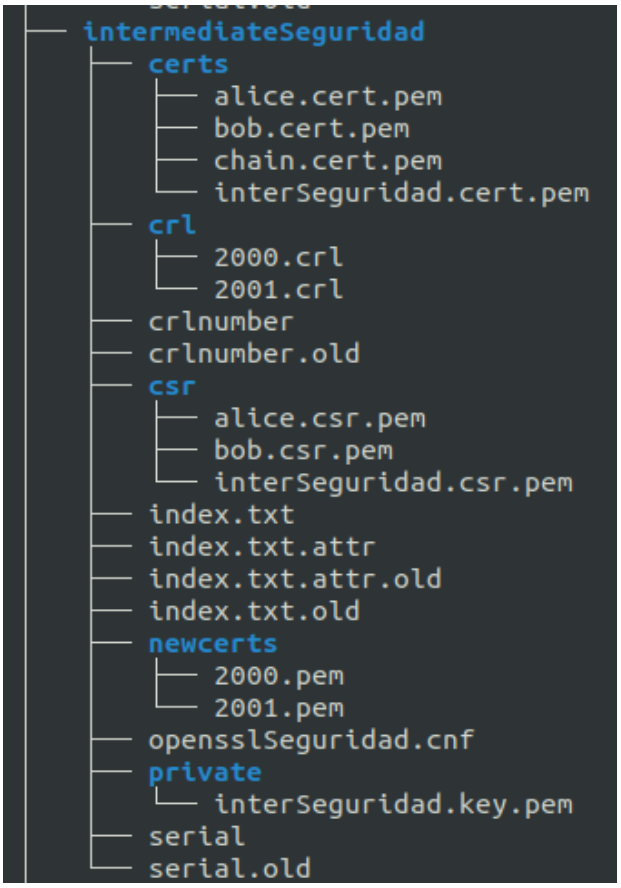


Figura 63: Estado del Esquema de certificados tras revokeCertsCA.sh

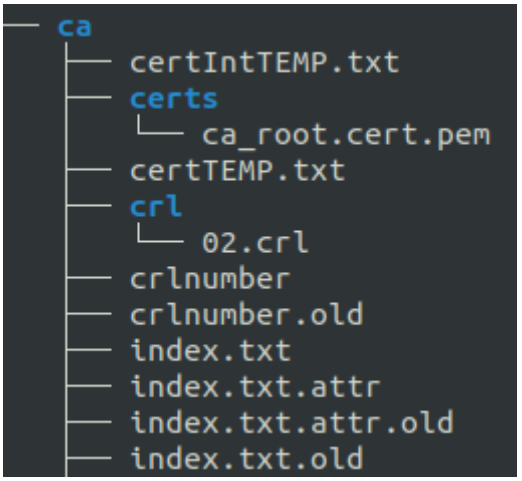


Figura 64: Estado del Esquema de certificados tras revokeCertsCA.sh

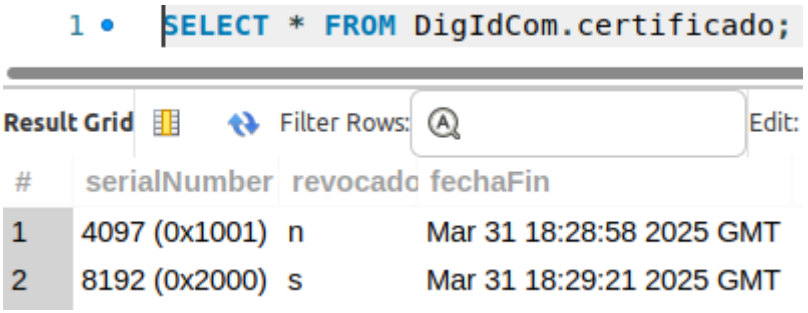


Figura 65: Estado de la tabla certificado de la BD tras revokeCertsCA.sh

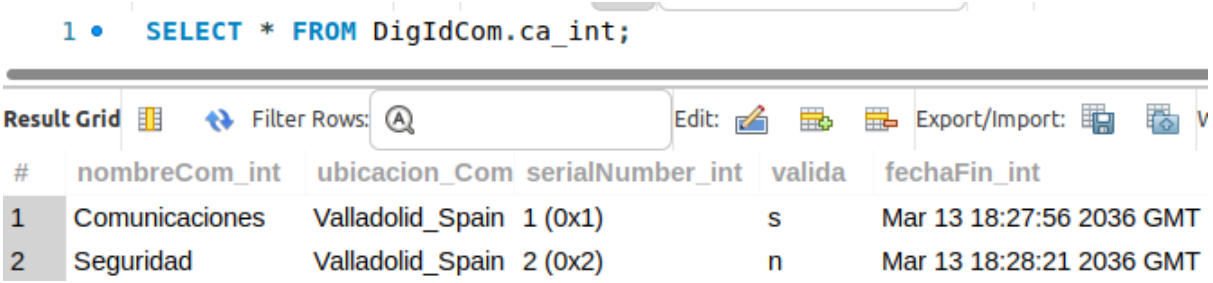


Figura 66: Estado de la tabla ca_int de la BDtras revokeCertsCA.sh

Y ahora realizamos la eliminación de la base de datos, para que quede eliminado de la representación actual del sistema.

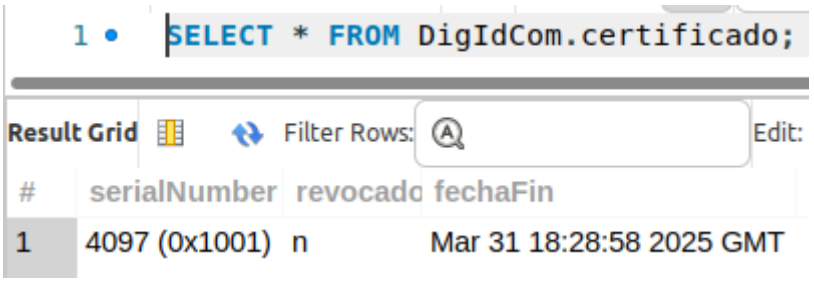


Figura 67: Estado de la tabla certificado de la BD tras dropCertCA.sh

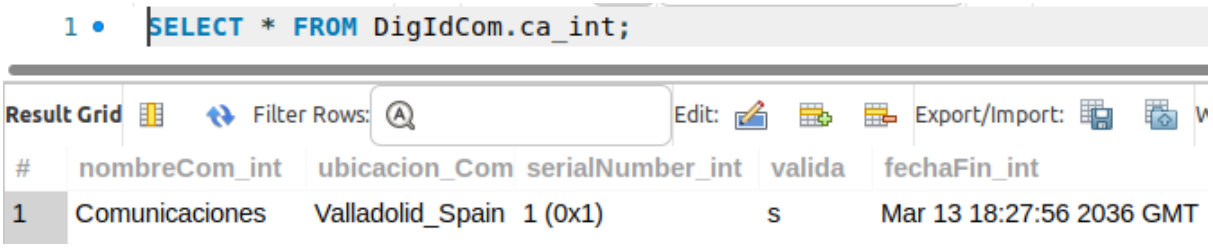


Figura 68: Estado de la tabla ca_int de la BD tras dropCertCA.sh

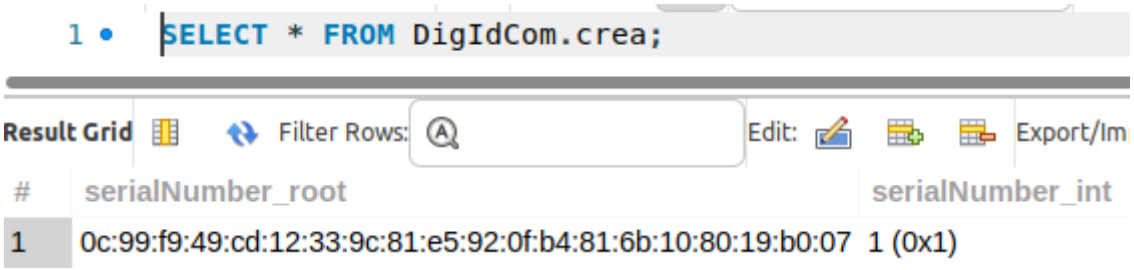


Figura 69: Estado de la tabla crea de la BD tras dropCertCA.sh

4.2 Funciones auxiliares

Los scripts que mostraremos a continuación verdaderamente solo trabajan contra la base de datos y no son más que la posibilidad de lanzar consultas desde la terminal de comandos. Como ya hemos explicado en numerosas ocasiones los scripts creados no automatizan el uso del sistema solo lo facilitan y por tanto es necesario un conocimiento del mismo es por eso que el uso de los siguientes scripts facilitan esto.

- **Consultar certificados de una comunidad**

1. sh certCom.sh <Comunidad> <gestor encargado>

Este script realiza una consulta contra la base de datos que permite conocer todos los certificados que pertenecen a una determinada comunidad. La consulta que emplea es:

SELECT serialNumber FROM emite WHERE serialNumber_int =
(SELECT serialNumber_int FROM certifica WHERE nombreComunidad = '$COMMUNITY');

- **Consultar certificados activos de una comunidad**

1. sh certActCom.sh <Comunidad> <gestor encargado>

Este script permitirá lanzar una consulta contra la base de datos que nos informa de cuáles son los certificados activos de una comunidad. El script realiza la siguiente consulta.

*SELECT * FROM DigIdCom.es_propietario_deWHERE serialNumber=*
ANY(SELECT serialNumber from DigIdCom.emiteWHERE serialNumber_int=
(SELECT serialNumber_int FROM DigIdCom.certifica WHERE nombreComunidad = '$COMMUNITY'))

AND

serialNumber = ANY (SELECT serialNumber from DigIdCom.certificado WHERE revocado ='n');

- **Consultar certificados de un miembro**

1. sh certsMember.sh <ID del miembro> < gestor encargado>

Como en los anteriores casos se lanzará una consulta que permite conocer cuales son todos los certificados que posee un miembro. La consulta que emplea es:

SELECT serialNumber,revocado FROM DigIdCom.certificado WHERE serialNumber =
ANY (SELECT serialNumber FROM DigIdCom.es_propietario_de WHERE ID = '$MEMBER');

- **Consultar certificados activos de un miembro**

1. sh certsActMember.sh <ID del miembro> < gestor encargado>

Para obtener los certificados activos de un miembro la consulta que este script ejecuta es:

SELECT nombreComunidad FROM DigIdCom.certifica WHERE serialNumber_int =
ANY (SELECT serialNumber_int FROM DigIdCom.emite WHERE (serialNumber =
ANY (SELECT serialNumber FROM DigIdCom.certificado WHERE revocado = 'n'))

AND

(serialNumber = ANY (SELECT serialNumber FROM DigIdCom.es_propietario_de WHERE ID = '$MEMBER')));

- **Consultar certificados revocados de un miembro**

1. sh certsRevocadosMember.sh <ID del miembro> < gestor encargado>

La consulta emitida por este script nos devuelve cuales son los certificados revocados de un miembro:

SELECT nombreComunidad FROM DigIdCom.certifica WHERE serialNumber_int =
ANY (SELECT serialNumber_int FROM DigIdCom.emite WHERE (serialNumber =
ANY (SELECT serialNumber FROM DigIdCom.certificado WHERE revocado = 's'))

AND

(serialNumber = ANY (SELECT serialNumber FROM DigIdCom.es_propietario_de WHERE ID = '$MEMBER')));

Capítulo 7 - Conclusiones y líneas futuras

En este proyecto se ha desarrollado un sistema de gestión de identidad para comunidades de usuarios. El objetivo principal se ha conseguido. Se ha comprobado que para dar satisfacción a los requisitos planteados era suficiente utilizar una infraestructura de clave pública basada en un OpenSSL, un sistema que como su propio nombre indica es abierto. Consideramos que esto es una ventaja. Hemos logrado crear un sistema funcional capaz de gestionar la identidad de los distintos miembros de una organización y hemos logrado realizar una incursión dentro de la gestión de identidad digital que sirve como un muy buen paso de partida para continuar investigando y encontrar nuevas vías en este campo.

El mundo de la gestión de identidades a día de hoy es un campo de estudio muy amplio y con gran capacidad de crecimiento en el futuro, proyectos nuevos surgen todos los días, como lo es el metaverso, hacen imperiosa la necesidad de afirmar que alguien es quien dice ser. Este proyecto a nivel personal me ha resultado muy útil para poder mostrarme gran parte de este mundo informático y que no solo limita a las líneas del mundo software tradicional que aunque naturalmente resulta imprescindible muchas veces es capaz de eclipsar el gran mundo que hay detrás. Considero que este proyecto me ha ayudado a comenzar a comprender una de las ramas que a nivel profesional y personal más me atraen en el mundo de la informática.

Durante el desarrollo de este proyecto hemos podido encontrar ciertas limitaciones técnicas, en aspectos principalmente derivados de la fase de análisis y del diseño de modelo de datos, ya que como hemos comentado al estar frente a un sistema en el que se podrían incorporar gran infinidad de funciones, el encontrarnos ante esta situación limitó mi capacidad de lograr enfocarme en una solución, que finalmente es la que se presenta en este documento. El lograr expresar en términos de modelado algo como un sistema, que podríamos considerarlo de un caracter más abstracto que un desarrollo software al uso, también resultó en un inicio complejo.

Una limitación del trabajo realizado es la falta de una interfaz gráfica para el control del sistema. Esto se plantea como una posible ampliación dentro del Trabajo Futuro.

Por último nunca me había visto en la necesidad de generar unos script shell con excesiva complejidad, por lo que en un inicio ciertos de los aspectos relacionados con esto se volvieron algo complicado, pero una vez comprendidas las estructuras más básicas con diversas pruebas, como en cualquier lenguaje de programación, esta tarea se volvió algo más sencilla lo que pudo derivar en la finalización de este proyecto. Poder comprender el uso de este tipo de scripts ayudó mucho en la integración de las dos partes de nuestro sistema ya que las herramientas de cada una de las partes son controlables con la ayuda de distintos comandos.

1. Líneas futuras

El objetivo de este último apartado es definir cuáles serían los siguientes pasos para nuestro sistema, para mi principalmente existen dos líneas de trabajo que harían mucho más interesante nuestro sistema

1. **Desarrollar una interfaz gráfica**

   Este apartado verdaderamente ya lo hemos tratado pero considero que el hecho de poder desarrollar una interfaz gráfica facilita un poco el uso del sistema, esto no haría que ya no fuese necesario el conocimiento del sistema dentro de la organización a la cual está dando servicio, pero sí reduciría tiempos en ciertas actividades que se realizan y sería algo más sencillo su uso.

2. **Ampliar la capacidad de soporte a más de una organización**

   Actualmente nuestro sistema únicamente permite la gestión de la identidad digital de una sola organización, sería muy interesante dotar nuestro sistema con esta capacidad, puesto que nos permitiría adaptarlo con mayor facilidad a situaciones más variadas del día a día.

3. **Añadir soporte para jerarquías intermedias**

   Si logramos dar soporte a jerarquías intermedias dentro de las comunidades de una organización, podríamos obtener mucha más precisión en las identidades que generamos para los usuarios de nuestro sistema, lo que permitiría al posible sistema IAM (*Identity and Access Management*) que acompañe al nuestro poder realizar un trabajo más simple y específico de los recursos que asigna.

4. **Añadir las funciones suficientes para convertir el sistema en un sistemas IAM**

   Como ya explicamos al inicio nuestro objetivo era el desarrollo de un sistema IDM (*Identity Management*) y con eso logrado el siguiente paso natural para nuestro sistema sería ganar las funcionalidades suficientes para poder ser catalogado como IAM. Para ello debemos de introducir las funcionalidades suficientes que permitan realizar la gestión del acceso a los diferentes recursos que la organización, donde esté implantado el sistema, tenga.

Bibliografía

[1] Asale, R. (s. f.). *criptografía | Diccionario de la lengua española*. «Diccionario de la lengua española» - Edición del Tricentenario. Recuperado 2 de abril de 2022, de https://dle.rae.es/criptograf%C3%ADa?m=30_2

[2] *Ciberseguridad y la identidad online de tu empresa*. (2021, 12 abril). INCIBE. Recuperado 22 de febrero de 2022, de https://www.incibe.es/protege-tu-empresa/blog/ciberseguridad-y-identidad-online-tu-empresa

[3] Córdoba, D. (2018, 2 febrero). *x509: Certificados digitales y codificaciones DER, CRT y CER*. Junco TIC. Recuperado 12 de marzo de 2022, de https://juncotic.com/x509-certificados-digitales-der-crt-cer/

[4] *Criptografía de clave pública - Glosario | MDN*. (2020, 8 diciembre). MDN. Recuperado 23 de febrero de 2022, de https://developer.mozilla.org/es/docs/Glossary/Public-key_cryptography

[5] DocuSign. (2020, 30 octubre). *¿Qué es la infraestructura de clave pública o PKI cuál es su relación con la firma electrónica?* Recuperado 23 de febrero de 2022, de https://www.docusign.mx/blog/pki

[6] González, A. (2021, 4 marzo). *Todo sobre la gestión de identidades o ID Management*. Ayuda Ley Protección Datos. Recuperado 22 de febrero de 2022, de https://ayudaleyprotecciondatos.es/2021/03/04/gestion-identidades/

[7] Google. (2017, 23 febrero). *Announcing the first SHA1 collision*. Google Online Security Blog. Recuperado 3 de junio de 2022, de https://security.googleblog.com/2017/02/announcing-first-sha1-collision.html

[8] Google & CWI Amsterdam. (2017, 27 febrero). *CWI and Google announce first collision for Industry Security Standard SHA-1*. CWI. Recuperado 2 de junio de 2022, de https://www.cwi.nl/news/2017/cwi-and-google-announce-first-collision-for-industry-security-standard-sha-1

[9] IBM. (2021, 20 abril). *Infraestructura de claves públicas (PKI)*. © Copyright IBM Corp. 2018. Recuperado 23 de febrero de 2022, de https://www.ibm.com/docs/es/ibm-mq/7.5?topic=ssfksj-7-5-0-com-ibm-mq-sec-doc-q009900--htm

[10] Institute of Electrical and Electronics Engineers, IEEE Computer Society. Standards Coordinating Committee, Institute of Electrical and Electronics Engineers, & IEEE Standards Board. (1990). IEEE Standard Glossary of Software Engineering Terminology. En . (pp. 1–84). IEEE.

[11] OpenSSL Foundation, Inc. (s. f.). *Manpages for 1.1.1*. OpenSSL. Recuperado 1 de marzo de 2022, de https://www.openssl.org/docs/man1.1.1/

[12] *¿Qué es el cifrado RSA y cómo funciona?* (s. f.). Ciberseguridad. Recuperado 25 de mayo de 2022, de https://ciberseguridad.com/guias/prevencion-proteccion/criptografia/cifrado-rsa/

[13] *RFC 5280 - Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*. (2008, mayo). RFC 5280. Recuperado 12 de marzo de 2022, de https://datatracker.ietf.org/doc/html/rfc5280

[14] Rus, C. (2017, 10 noviembre). *Cuando una librería de códigos pone en peligro la identidad de media nación: así se ha quebrantado el. . .* Xataka. Recuperado 30 de junio de 2022, de https://www.xataka.com/seguridad/cuando-un-algoritmo-pone-en-peligro-la-identidad-de-media-nacion-asi-se-ha-quebrantado-el-cifrado-rsa-2048

[15] Segovia, E. (2021, 19 noviembre). *¿Qué es una Autoridad de Certificación?* Ivnosys Soluciones. Recuperado 17 de marzo de 2022, de https://www.ivnosys.com/es/que-es-una-autoridad-de-certificacion/

[16] *Simple PKI — OpenSSL PKI Tutorial*. (s. f.). Pki-Tutorial.Readthedoc. Recuperado 16 de febrero de 2022, de https://pki-tutorial.readthedocs.io/en/latest/simple/#overview

[17] Tecnozero Soluciones Informáticas. (2021, 13 marzo). *«The Heartbleed Bug»: el fallo de seguridad que sacude Internet.* Recuperado 30 de mayo de 2022, de https://www.tecnozero.com/blog/heartbleed-bug/

[18] Tilborg, H. V. C. A., & Jajodia, S. (2011). *Encyclopedia of Cryptography and Security* (2.a ed.). Springer.

[19] *Vulnerabilidad Heartbleed*. (2020, 17 julio). INCIBE-CERT. Recuperado 30 de mayo de 2022, de https://www.incibe-cert.es/alerta-temprana/bitacora-ciberseguridad/vulnerabilidad-heartbleed