PROYECTO FINAL DE CARRERA

DESARROLLO EN PYTHON DE UNA PLATAFORMA PARA LA REVOCACIÓN DE CERTIFICADOS DIGITALES EN VANETS

Estudios : Ingeniería de Telecomuniación

Autor : Francisco Ronaldo Medina Huerta

Director : José Luis Muñoz Tapia

Año : 2010

Agradecimientos

Quiero aprovechar estas líneas para dar gracias a todos los que han colaborado, tanto directa como indirectamente, en la realización de este proyecto.

En primer lugar, quiero dar gracias a Carlos por haberme orientado siempre en todos los detalles y dudas que me han surgido a lo largo de la realización de este proyecto. Gracias a José Luis por darme esta oportunidad y por su apoyo en este proyecto.

Gracias a mis amigos por haber estado siempre a mi lado durante esta etapa tan importante de mi vida, ya sea desde aquí Barcelona o desde la distancia: a Valeria, Carlos, Bite, Renato, Christian, Rosa y Jaime en Perú; y aquí en Barcelona a Silvia, Gustavo, Renzo, Ronald, Natali, Julio, Javi y de manera especial a Gemma. Vuestro apoyo ha sido fundamental para llegar donde estoy ahora.

Finalmente, no me caben palabras para expresar el eterno agradecimiento hacia mi familia: a mis padres, Francisco y Nery, y a mi hermana Paloma, por la comprensión y el apoyo incondicional que siempre me habéis brindado. Gracias por haber estado siempre y por seguir creyendo en mí, ya que sin vosotros nada de esto hubiera sido posible.

A todos vosotros, y a quien haya podido olvidar, ¡muchas gracias!

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

GLOSARIO DE ACRÓNIMOS

3-DES	Triple Digital Encryption Standard
AES	Advanced Encryption Standard
BS	Base Station
CA	Certification Authority
CP	Certification Policy
CRI	Constant Risk Issuance
CRL	Certificate Revocation List
CRS	Certificate Revocation System
CSI	Certificate Status Information
DES	Digital Encryption Standard
DN	Distinguished Name
DSS	Digital Signature Standard
DTN	Disruption Tolerant Network
EDI	Revocation Data Issuer
H-CRS	Hierarchical Certificate Revocation System
MANET	Mobile Ad-hoc Network
MD5	Message Digest 5
MHT	Merkle Hash Tree
MN	Mobile Node
OCSP	Online Certificate Status Protocol
PGP	Pretty Good Privacy
PKI	Public Key Infrastructure

PKC	Public Key Cryptography
RSA	Rivest Shamir Adleman
SET	Secure Electronic Transactions
SLCH	Skip Lists with Commutative Hashing
SHA-1	Security Hash Algorithm
SPKI	Simple Public Key Infrastructure
TTP	Trusted Third-Party
X.509 IC	X.509 Identity Certificate

Capítulo 1

INTRODUCCIÓN

1.1 Introducción

Actualmente vivimos en una Sociedad de la Información en la cual la seguridad en los sistemas informáticos se ha convertido en una de las tareas más importantes desde la aparición y expansión de Internet.

Los certificados digitales (X.509 PKIX, RFC 5208 **[1]**) representan un punto muy importante en las transacciones electrónicas seguras. Estos brindan una forma conveniente y fácil de asegurar que los participantes de una transacción puedan confiar el uno con el otro. Dicha confianza se establece a través de un tercero llamado Autoridad de Certificación.

Sin embargo, muchas veces un certificado digital emitido puede dejar de ser seguro, al caer comprometida su clave privada y dejar de ser una fuente confiable. Para ello existe el proceso de Revocación de un certificado digital.

La Revocación consiste en la anulación de la vigencia de un certificado digital antes de la fecha de expiración especificada en él. Puede ser solicitada en cualquier momento, especialmente si es que su seguridad y validez puedan haber sido comprometidas.

Dentro de los sistemas de comunicaciones, las redes vehiculares emergen como una de las instancias más convincente a futuro, así como una de las que

más retos tienen por superar. Además de la dificultad de su despliegue, la seguridad es un factor crítico y muy importante aún por vencer.

Validar el estado de un certificado digital es un proceso muy importante que se vuelve sumamente crítico en un entorno móvil. La validación de certificados se hace mucho más compleja en Redes Ad-hoc Vehiculares (VANETs) debido especialmente a que la continuidad de la conexión punto a punto entre el cliente y servidor no está siempre garantizada.

Actualmente una amplia gama de mecanismos de validación de certificados son utilizados. Estos mecanismos usan infraestructuras para manejar la distribución del estado actual del estado de revocación de un certificado hacia el cliente. Sin embargo, en entornos de baja conectividad como un entorno móvil, estos usuarios requieren de una solución para soportar la posible pérdida de conectividad con las autoridades confiables que están ubicadas dentro de la infraestructura de la red.

Además, la aceptación de un certificado requiere la validación del mismo en tiempo real, esto es, cuando un certificado en particular está a punto de ser utilizado. El problema ocurre que en una VANET los usuarios podrían estar desconectados del Servidor cuando la validación sea necesaria.

1.2 Motivación

La principal motivación de este proyecto viene dada debido al problema descrito en la introducción de este proyecto: Debemos buscar una manera de mantener informado al usuario de la “lista negra” de usuarios cuya seguridad de su certificado digital haya sido comprometida. Esto es, aquellos con los cuales ya no será confiable realizar una transacción electrónica de cualquier tipo.

Al estar en un entorno en el cual las desconexiones del usuario con el servidor son muy comunes, al menos debemos asegurarnos que si el cliente va a interactuar con el certificado de otro cliente, lo haga de la forma más segura, y teniendo la información más actual posible para determinar si el certificado de dicho usuario es confiable o no.

En este sentido, en la referencia **[15]** se propone un mecanismo basado en el punto de vista del propio usuario, desarrollado por los directores de este proyecto. Este mecanismo permite calcular el riesgo que se asume al usar un certificado cuya validez está en duda debido a que el cliente se encuentra en un estado de desconexión y no puede solicitar al servidor la descarga de la lista de certificados revocados más actual. El riesgo es un valor que indica la probabilidad que existe de dar por válido un certificado cuando realmente este se encuentra revocado. Lo que se requiere es un Sistema de Revocación de Certificados que sea capaz de evaluar el riesgo real que existe en el sistema, ya que en el mecanismo propuesto se calcula este valor usando métodos estadísticos.

Otra de las motivaciones del proyecto es el desarrollo de una aplicación en el lenguaje de programación Python[1], el cual es una tecnología que está muy de moda en estos últimos tiempos y el cual, por su simplicidad y potencia, permite realizar aplicaciones de una manera rápida, sencilla y óptima para su despliegue en distintas plataformas.

Python posee una sintaxis clara y sencilla, permite el tipado de datos dinámico, gestión de memoria, y además existen una gran cantidad de librerías disponibles para apoyar en el desarrollo con este lenguaje. Muchos son los casos de grandes empresas que aplican Python en sus aplicaciones con gran éxito, tal es el caso de Google, Yahoo o la NASA, por nombrar algunas.

Al ser Python una tecnología emergente con amplia distribución, considero que es un gran aporte para el aprendizaje durante el desarrollo de este proyecto.

1.3 Objetivos

El proyecto tiene como principal objetivo la implementación de un Servidor de Revocaciones de Certificados Digitales, el cual sea capaz de recibir principalmente peticiones de distintos clientes en las cuales se puede requerir:

- La Revocación del Certificado digital de un cliente.
- El envío de una CRL a un cliente específico, el cual contendrá la lista de certificados revocados en un momento dado.

[1] **Python Home:** http://python.org

Para esto, la Aplicación a desarrollar se basará en una Plataforma ya existente, denominada Cervantes (Certificate Validation Test-Bed)[2] **[2]**, la cual está desarrollada en Java y presenta las funcionalidades antes descritas.

Cabe resaltar que la Aplicación actual ha sido desarrollada en Python, tomando únicamente como referencia la documentación elaborada en el desarrollo de Cervantes, debido a que la arquitectura y despliegue del Sistema se ha optimizado para el trabajo con Python.

El otro objetivo fundamental de este proyecto es el estudio y evaluación del Riesgo, un nuevo mecanismo basado en el usuario que permite determinar la aceptación o rechazo de un certificado cuando el cliente se encuentra en estado de desconexión y su lista de certificados revocados está desactualizada. El objetivo en sí es preparar al servidor para ser capaz de tomar todas las estadísticas necesarias para poder calcular el riesgo real existente en el sistema.

Adicionalmente, otro objetivo para el desarrollo de este proyecto es simular el entorno de los clientes que van a enviar sus solicitudes al Servidor de Revocaciones, como una MANET, donde a todos los usuarios se les añade el concepto de movilidad (a través de dos algoritmos distintos que permiten modelar el movimiento de los nodos dentro de una MANET) y de interrupciones, esto último, para ponernos en el caso de una red VANET, donde las desconexiones de los clientes son muy frecuentes. Para esto último simplemente se tendrá un escenario con una zona de cobertura definida.

Cuando el usuario esté fuera de dicha zona de cobertura, no podrá conectarse con el servidor, por lo que deberá hacer uso de los datos almacenados en caché para evaluar un certificado específico.

[2] **Cervantes Home**: http://isg.upc.es/cervantes

1.4 Estructura de la Memoria

La estructura del presente proyecto es la siguiente:

- En el Capítulo 2,"Estado del Arte", se presentarán rápidamente algunos conceptos básicos de criptografía, seguridad, certificados digitales, redes VANET y redes DTN que permitirán entender completamente el desarrollo del proyecto. Además se hablará de las tecnologías informáticas utilizadas para la implementación del sistema de revocación.

- En el Capítulo 3, "La Revocación de Certificados", se hará una descripción del paradigma de la Revocación de Certificados Digitales, detallando los elementos y mecanismos involucrados en el proceso de revocación, y se detallará el concepto del *Riesgo* y el uso de un mecanismo basado en el punto de vista del usuario, para emtir CRLs manteniendo un valor de riesgo (considerar un certificado revocado como válido) constante.

- En el Capítulo 4, "Desarrollo de la Plataforma", haremos una descripción del Sistema de Revocación de Certificados Digitales implementado. Este sistema es una aplicación Cliente/Servidor desarrollada en Python, a lo largo de este proyecto, para la migración del antiguo sistema Cervantes existente, y para el test y evaluación del Riesgo del uso de una CRL en el entorno de una red VANET.

- En el capítulo 5, "Simulación y Evaluación de Resultados", haremos un análisis y evaluación de los resultados obtenidos durante distintos casos de simulación, donde variaremos parámetros e iremos comprobando el comportamiento de la aplicación a través de gráficas desarrolladas en Matlab.

- En el capítulo 6, "Conclusiones y Líneas Futuras" se presentarán las conclusiones más relevantes obtenidas de la implementación de este proyecto, además de presentar líneas de guía para futuros desarrollos e implementaciones dentro de la línea de este proyecto.

Capítulo 2

ESTADO DEL ARTE

2.1 Criptografía: Conceptos Básicos

Se puede definir la criptografía como la ciencia que permite cifrar y descifrar los mensajes para que resulte imposible conocer su contenido a los que no dispongan de unas claves determinadas. En el campo de la informática el uso de la criptografía es muy habitual, siendo utilizado en las comunicaciones seguras y en el almacenamiento de información.

En las comunicaciones, se utiliza algún algoritmo criptográfico que permite alterar mediante una clave secreta la información a transmitir, de manera que este mensaje viaja cifrado hasta llegar a su punto de destino, donde un sistema que conoce la clave de cifrado es capaz de descifrar la información contenida en el mensaje y volverla inteligible.

Incluso si un atacante consiguiera interceptar los mensajes enviados, este todavía no sería capaz de leer los datos si es que estos están protegidos mediante algún algoritmo criptográfico o de cifrado.

Sin embargo, además de ser utilizada para ocultar el significado de los datos, la criptografía realiza otras necesidades críticas de seguridad para la transmisión de datos, entre ellas:

- Autentificar que el remitente de un mensaje es el remitente real y no un impostor. Además permite probar que un usuario ha llegado a enviar un mensaje o realizar una acción determinada. En efecto, permite la generación de Certificados Digitales, sobre los cuales se basa el esquema de seguridad de una Infraestructura de Clave Pública (PKI) **[3]**.
- Respetar la confidencialidad, ya que sólo un lector con el algoritmo de descifrado correcto puede leer el mensaje cifrado.
- Proteger la integridad de la información, garantizando que los mensajes enviados no han sido alterados durante la transmisión.

2.1.1 Criptografía Simétrica

Este tipo de criptografía se basa en el empleo de algoritmos simétricos que usan una misma clave para encriptar y desencriptar los mensajes. Este tipo de criptografía es denominado a veces Criptografía de Clave Secreta.

La encriptación Simétrica implica el uso de una única llave *K* para encriptar y desencriptar los datos, de manera que tanto el emisor como el receptor de la información deben compartir el conocimiento de la clave secreta que es utilizada para encriptar y desencriptar los mensajes intercambiados entre ambos.

FORMALMENTE. *El mensaje M es encriptado aplicando el algoritmo simétrico S sobre M utilizando la clave K:*

$$C = S_K(M)$$

El mensaje secreto C es desencriptado aplicando el algoritmo inverso S^{-1} *al mensaje secreto C utilizando la clave K:*

$$M \triangleq S_K^{-1}(C)$$

Durante muchos años el algoritmo DES (Digital Encryption Standard) y su versión extendida, Triple-DES (3-DES) han sido los algoritmos de clave

simétrica más populares. Recientemente, el AES (Advanced Encryption Standard) se ha convertido en el sucesor del DES.

Los algoritmos de clave simétrica son en general, más sencillos y rápidos que los de clave asimétrica, pero su principal desventaja es que se requiere que de alguna manera las dos partes de la comunicación puedan intercambiar la clave simétrica en una forma segura. Este problema es aún más relevante en escenarios grandes y es denominado “Problema de distribución de claves”.

2.1.2 Criptografía Asimétrica

Este tipo de criptografía se basa en el empleo de algoritmos de clave pública que usan distintas claves para encriptar y desencriptar los mensajes. Es también llamada Criptografía de Clave pública (PKC). Fue inventada en el año 1976 por los matemáticos Whit Diffie y Martin Hellman y es la base de la criptografía moderna.

La criptografía asimétrica utiliza dos claves complementarias llamadas Clave Privada y Clave Pública. Los mensajes que son encriptados con una clave privada necesitan de su correspondiente clave pública para ser desencriptados. Y viceversa, los mensajes encriptados con una clave pública sólo pueden ser desencriptados con su clave privada.

Las claves privadas deben ser conocidas únicamente por su propietario, mientras que la correspondiente clave pública puede ser dada a conocer abiertamente. Esto permite garantizar la identidad del emisor y del receptor de la información.

Este tipo de criptografía está basada en la utilización de números primos muy grandes. Al multiplicar entre sí dos números primos muy grandes, el resultado obtenido no puede descomponerse eficazmente. Es decir, incluso utilizando ordenadores muy avanzados y métodos aritméticos muy avanzados, sería casi imposible. Cabe resaltar que este proceso será más seguro cuanto mayor sea el tamaño de los números primos utilizados.

Algunos protocolos actuales, como SET o PGP utilizan claves generadas con números primos de un tamaño tal que los hace completamente sólidos e inexpugnables.

La desventaja del uso de algoritmos de claves simétricas es que cuando el mensaje a tratar es muy largo, el proceso de encriptación es muy lento.

FORMALMENTE.

1. *Es computacionalmente fácil para una parte **B** generar un par: (clave pública KU_S, clave privada KR_B).*
2. *Es computacionalmente fácil para un emisor **A**, conociendo la clave pública y el mensaje a encriptar M, generar el correspondiente texto cifrado:*

$$C = E_{KU_B}(M)$$

3. *Es computacionalmente fácil para un receptor **B** desencriptar el texto cifrado resultante **C** utilizando su clave privada para recuperar el mensaje original*

$$M = D_{KR_B}(C) = D_{KR_B}[E_{KU_B}(M)]$$

4. *Es computacionalmente imposible para un oponente, conociendo la clave pública KU_B, determinar la clave privada KR_B.*
5. *Es computacionalmente imposible para un oponente, conociendo la clave pública, y el texto cifrado **C**, recuperar el mensaje original **M.***

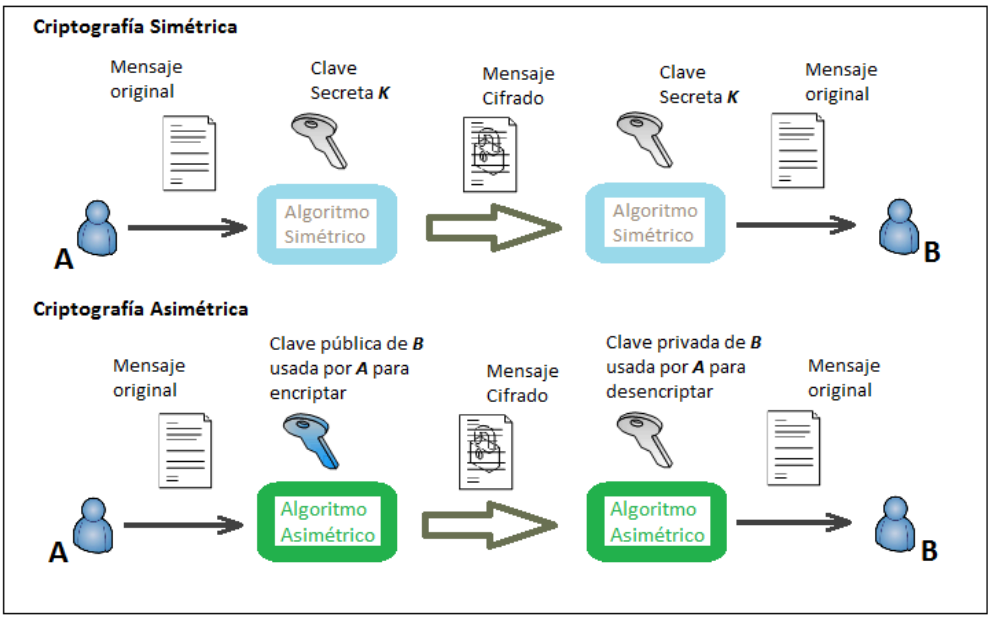


Figura 2.1. Criptografía Simétrica y Asimétrica.

Los sistemas de Clave Pública son principalmente utilizados para los siguientes propósitos:

- **Encriptación:** el emisor encripta un mensaje con la clave pública del receptor.

- **Firma Digital:** emula una firma real y física generando una prueba digital que solo el creador/emisor del mensaje puede crear, pero que todos los demás pueden identificar como perteneciente al creador. La encriptación con una clave privada del creador sirve como una firma que solo el dueño de la clave privada puede crear, pero que cualquiera con la clave pública puede verificar.

- **Intercambio de claves:** Dos partes cooperan para intercambiar una clave de sesión (clave simétrica). Algunos protocolos modernos utilizan un algoritmo de encriptación simétrico para encriptar un mensaje el cual esta previamente encriptado con una clave, y por otro lado intercambian esta clave encriptada mediante un algoritmo simétrico.

Los algoritmos de clave pública más conocidos son el RSA y DSS.

2.2 El Certificado Digital

Los certificados pueden ser definidos como una forma de distribuir las claves públicas, debido a que son firmados digitalmente por una Entidad Externa Confiable (TTP, Trusted Third-Party) denominada Emisor de Certificados (Certificate Issuer).

Un Certificado Digital es emitido por esta entidad, la cual es llamada Autoridad de Certificación (CA, Certificación Authority). El certificado emitido garantiza que la CA ha verificado y confía plenamente en la identidad de la persona a la que pertenece. La CA manifiesta esta conformidad firmando el certificado y adjuntando dicha firma al final del mismo certificado.

Esta firma es efectuada con la clave privada de la CA que sólo ella conoce. Esto permite que cualquier receptor del certificado digital pueda verificar, utilizando la clave pública de la CA, que el certificado ha sido firmado por esta. Otro documento que suele firmar la CA es la Lista de Revocación de Certificados (CRL), la cual será detallada en el siguiente capítulo.

De tal manera, podemos ver que el Certificado asocia una única clave pública a una persona. Esta clave se adjunta con el contenido del certificado junto con otros parámetros de la clave. En el proceso del firmado digital se utiliza el certificado digital para constatar que la clave pública, necesaria para la verificación de la firma, pertenece al firmante.

Existen varios tipos de certificados digitales, entre los cuales tenemos:

- X.509 ICs (Identity Certificate).
- Certificados SPKI
- Certificados PGP
- ACs (Attribute Certificates)

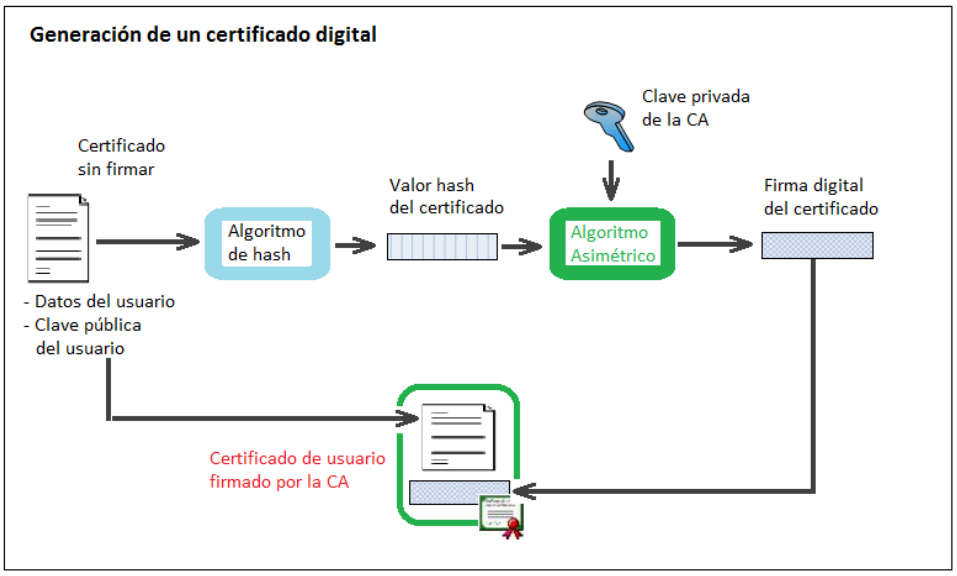


Figura 2.2. Generación de un Certificado Digital.

2.2.1 Certificados X.509

En el desarrollo de este proyecto, trabajaremos con los Certificados de Identidad X.509 (X.509 IC).

Los Certificados X.509 son uno de los tipos de certificados cuyo uso está más extendido a lo largo de Internet. Su principal función es la de asociar una clave pública con una identidad determinada.

En este sentido:

> *Un Certificado de Identidad establece una asociación entre un nombre, denominado **Distinguished Name (DN)** y la clave pública del usuario.*

Además, la autenticación de un certificado confía en que cada usuario tiene un único DN. Los DNs se definen bajo el estándar X.588 el cual define que estos serán únicos por cada certificado en todo Internet.

Actualmente, existen tres versiones de Certificados X.509:

- La versión 1 ha estado disponible desde 1988. Está ampliamente desplegada y es la más genérica.
- La versión 2 introduce el concepto de Identificadores Únicos para el Sujeto (Subject) y para el Emisor (Issuer) del certificado, de manera que se pueda manejar la posibilidad de reutilizar los nombres del Emisor y del Sujeto a lo largo del tiempo. Sin embargo, como la mayoría de certificados recomiendan no reutilizar estos nombres, esta versión no es muy utilizada.
- La versión 3 es la más reciente, apareció en 1996 y soporta la noción de extensiones, lo que permite definir información adicional e incluirla en el campo Extensión del cuerpo del certificado.

Para el desarrollo de este proyecto se ha considerado que la aplicación implementada solo soporta los Certificados X.509 versión 1.

Todos los Certificados X.509 deben tener la siguiente información, adicionalmente a la firma digital:

- **Version** (versión): Identifica la versión del estándar X.509 aplicado al certificado, lo cual afecta al tipo de información que pueda estar especificado en él.
- **Serial number** (número de serie): La entidad que crea el certificado es responsable de asignar un número de serie para distinguir a ese certificado de los otros que haya emitido dicha entidad.
- **Signature Algorithm** (algoritmo de la firma): Identifica el algoritmo asimétrico utilizado por el emisor para firmar el certificado.
- **Issuer Name** (Nombre del emisor): El DN del emisor.
- **Validity Period** (periodo de validez): Cada certificado es válido solo por un intervalo de tiempo. No es válido antes de la fecha de activación (not-valid-before) ni es válido posteriormente a la fecha de expiración (not-valid-after).
- **Subject Name**: El DN de la entidad a la cual la clave pública del certificado referencia.
- **Subject Public Key Information** (Información de Clave pública del Sujeto): Es la clave pública de la entidad que está siendo identificada en el Certificado, junto al identificador del algoritmo que especifica qué algoritmo de encriptación se utiliza para encriptar esta clave, además de cualquier otro parámetro o información que pueda estar asociado a la clave pública.

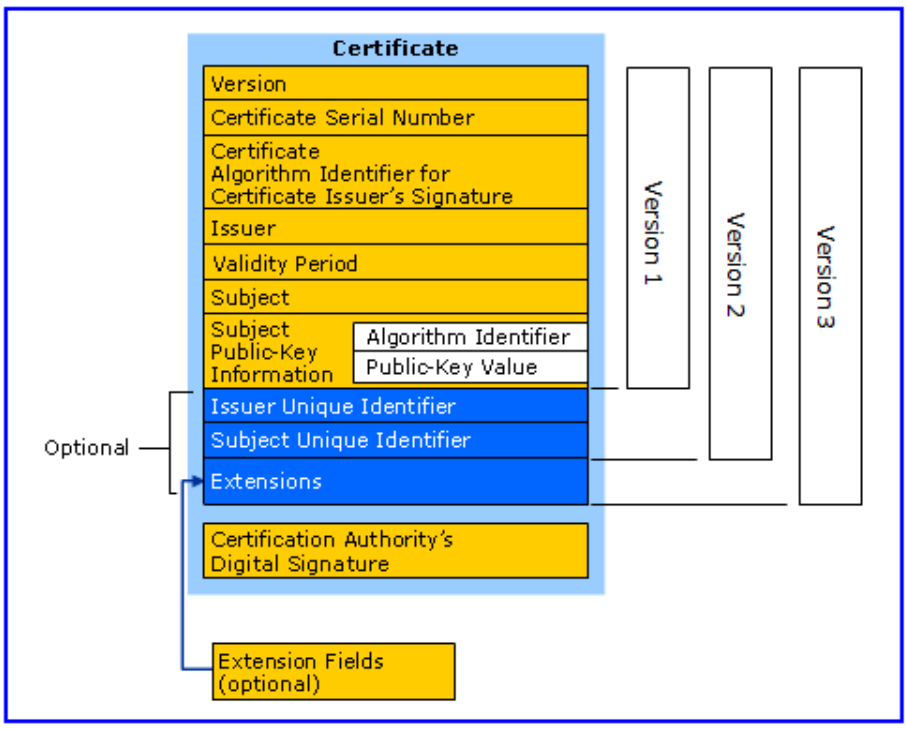


Figura 2.3. Estructura de Certificado X.509

2.2.2 Firma Digital

En el proceso de firmado digital los algoritmos hash o "hash codes" tienen un papel importante. Estos algoritmos son Sistemas Criptográficos de Resumen que aceptan de entrada un mensaje de longitud abierta y tras aplicar un algoritmo devuelven una cadena de bits de longitud fija. Esta longitud varía según el tipo de algoritmo que se use.

Los algoritmos hash más conocidos actualmente son los Message Digest (siendo el más usado el Message Digest 5 ó MD5 **[4]**) y los Secure Hash Algorithm (siendo los más usados el SHA-1 **[5]**).

Al valor obtenido luego de haber aplicado el algoritmo hash se le llama valor del hash, o simplemente hash, y tiene algunas características interesantes como que el valor que produce cada mensaje es único, que no puede haber otro mensaje que produzca el mismo valor y que es imposible reconstruir el mensaje a partir de su hash.

En el proceso de obtención de la firma digital de un mensaje, el involucrado posee dos pares de claves distintas que se llaman clave privada y clave pública, la clave privada se usa para encriptar y debe permanecer secreta mientras que la otra se usa para desencriptar y debe permanecer en un certificado digital. Además, se debe acordar en un algoritmo de firma y en un algoritmo hash que usarán tanto el firmante como la persona que verifique la firma.

El primer paso es obtener el valor hash del mensaje y en el segundo paso el firmante debe utilizar su clave privada para encriptar este valor hash, a estos dos pasos se les conoce como firmar digitalmente un mensaje y al resultado obtenido como firma digital.

La verificación de la firma digital implica que otra persona desencripte la firma digital utilizando la clave pública del firmante. Si la puede desencriptar correctamente puede confiar en que el firmante realmente firmó el mensaje utilizando su clave privada. El resultado de la verificación es lo que se encriptó con la clave privada, en este caso el valor hash del mensaje. Si además el verificador tiene una copia del mensaje original en su poder, puede calcular el valor hash del mensaje y compararlo con lo obtenido en la desencriptación. Si los valores son iguales entonces puede confiar en que el mensaje que tiene es el mismo que el que recibió firmado.

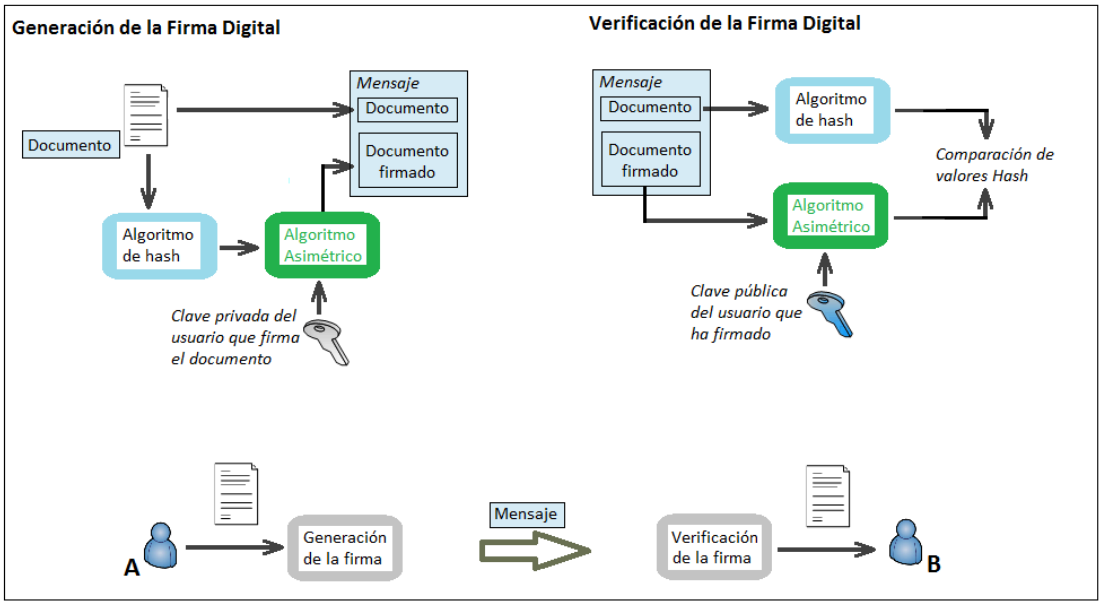


Figura 2.4. Generación y Verificación de Firma Digital.

2.3 Infraestructura de Clave Pública (PKI)

La infraestructura de clave pública (PKI) se puede definir como el conjunto de hardware, software, personas, políticas y procedimientos necesarios para crear, administrar, almacenar, distribuir y revocar certificados de clave pública basados en criptografía de clave pública **[1]**.

De manera que, para tratar con certificados, no solo la CA es necesaria, sino que también necesitaremos de esta infraestructura para asegurar la validez de las transacciones digitales usando certificados digitales.

2.3.1 Características de una PKI

Las infraestructuras de clave pública tienen en cuenta cuatro aspectos fundamentales en cualquier comunicación electrónica, estos son especialmente importantes si se habla de una transacción electrónica:

- **Confidencialidad**: Una comunicación entre dos personas no debe ser vista ni interferida por una tercera persona que no forme parte de la comunicación. La tecnología PKI usa la encriptación para asegurar la confidencialidad de los datos críticos que se encuentran en tránsito en la comunicación actual. También ofrece la posibilidad de encriptar datos valiosos almacenados en los servidores que tienen que conectarse a Internet, esto es importante porque si estos datos llegaran a ser interceptados, el atacante aún tendría que romper la encriptación antes de que pueda sacar ventaja de los mismos.

- **Autenticación**: Significa que el acceso a una comunicación electrónica debe estar restringido solo a quienes puedan presentar las credenciales necesarias de identidad. La forma más común de acreditar la identidad es a través de un identificador de usuario y una contraseña. Esta forma está considerada como un bajo nivel de autenticación y es frecuentemente fácil de romper. La tecnología PKI utiliza el certificado digital como credencial de identificación.

- **Integridad**: Los datos recibidos deben ser los mismos que los datos enviados, esto quiere decir que no deben haber sido modificados durante su transmisión ya sea por error o a propósito. La tecnología PKI utiliza los algoritmos hash para asegurar la integridad de estos datos. Las características de estos algoritmos hacen que cualquier cambio producido en el mensaje original durante su transmisión por la red se detecte como una pérdida de integridad. Normalmente se envía el mensaje y el valor de su hash, el receptor calcula el hash del mensaje recibido y lo compara con el valor del hash que le envió el transmisor, si los valores son idénticos se puede asegurar que el mensaje no ha sido modificado.

- **No-repudio**: Significa que si surge una discrepancia sobre lo que sucedió en una comunicación electrónica que implica intercambio de datos, habrá innegable evidencia presente dentro del sistema de comunicación que pueda ser utilizada para probar con suficiente certeza lo que realmente sucedió. Esto es especialmente sensible en operaciones que implican la firma de contratos electrónicos, en las cuales se evitaría que cualquiera de las partes que están implicadas en el contrato repudie lo que ha firmado. La tecnología PKI provee no-repudio a través del uso de la firma digital.

2.3.2 Funcionamiento básico de una PKI

Alice quiere comunicarse de manera segura con Bob. Esto quiere decir que Alice no quiere que nadie más escuche esta conversación, quiere que la información enviada a Bob no sea alterada durante su transmisión y finalmente le gustaría disponer de algún mecanismo que pueda probar que ellos tuvieron esta conversación, en caso que Bob por algún motivo quiera negarla.

En la siguiente figura describiremos los pasos básicos y la infraestructura necesaria para establecer esta comunicación segura entre Alice y Bob

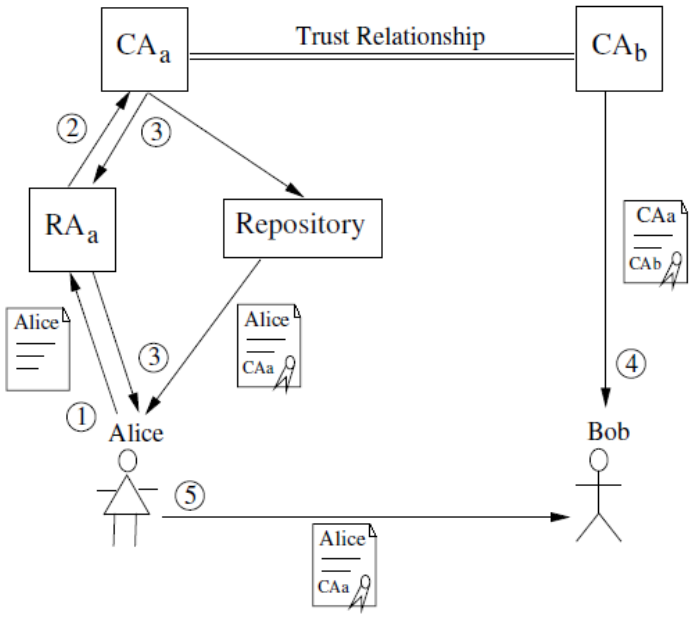


Figura 2.5. Uso de certificados en transacciones seguras

1. Alice crea una clave pública y una privada utilizando un algoritmo de clave pública. Luego, ella crea una solicitud de certificado, lo cual es el certificado justo antes de ser firmado por la Autoridad de Certificación. Alice envía su solicitud de certificado a la Autoridad de Registro (RA, Registration Authority) para ser firmado.

2. Cualquier acción de aprobación o desaprobación tiene lugar en la RA. Luego, la RA envía una solicitud a la CA para aprobación de políticas y para ser firmado.

3. El resultado de la firma del certificado es enviado de vuelta a Alice a través de la RA o es almacenada temporalmente en un repositorio.

4. Alice puede anunciar que su clave pública es confiable. Bob, que quiere comunicarse con ella, pregunta por su certificado. Bob, para poder verificar el certificado de Alice, obtiene la clave pública de la CA que firmó la clave pública de Alice.
   Él necesita hacerlo de manera segura. Si ambos están en la misma CA él ya tendrá la clave pública de dicha CA. En caso contrario, Bob solicitará a su CA que contacte a la CA de Alice para obtener su clave pública.

5. Finalmente, teniendo tanto Alice como Bob las claves públicas de ambos, pueden comunicarse de manera segura.

2.3.3 Infraestructura de Clave Pública X.509 (PKIX)

Como hemos visto hasta ahora, los certificados digitales son una parte fundamental de la tecnología PKI. Los esfuerzos por desarrollar una arquitectura basada en certificados sobre Internet llevaron a adoptar el modelo de arquitectura basado en los certificados X.509 desarrollado por el grupo de trabajo PKIX del IETF **[6]**.

Actualmente el término PKIX se refiere a la infraestructura de clave pública basada en certificados X.509 (*Public Key Infrastructure* X.509) y el término certificado PKIX usualmente hace referencia a los perfiles de certificado y de listas de revocación basados en el estándar de certificados X.509v3.

Los elementos clave que lo componen son:

- **Entidad final (End Entity)**: Es el nombre genérico que reciben los usuarios de una PKI o los dispositivos que forman parte de ella, como enrutadores o servidores. Estos pueden ser identificados en el campo que define al propietario del certificado X.509. Las entidades usuarios normalmente utilizan los servicios que ofrece la PKI y los dispositivos normalmente los soportan.
- **Autoridad de Certificación (CA)**: Es la autoridad encargada de emitir los certificados digitales X.509 y usualmente también las listas de revocación de certificados (CRLs), aunque a veces delega la función de emitir CRLs a un elemento denominado Emisor de CRL. También puede desempeñar funciones administrativas como las de registro de entidades finales o publicación de certificados, aunque normalmente estas funciones son desempeñadas por las autoridades de registro.
- **Autoridad de Registro (RA)**: Es un elemento opcional de la arquitectura PKIX que puede asumir algunas funciones administrativas de la CA, tal vez la más común sea el proceso de registro de las entidades finales, pero

también puede realizar otras funciones como el proceso de revocación de certificados y el manejo de los datos de la entidad final.

- **Emisor de CRLs (CRL Issuer)**: Es un elemento opcional de la arquitectura PKIX al que la CA puede delegar la función de emitir las CRLs. Algunas veces se encuentra integrado en la CA a modo de servicio.
- **Repositorio (Repository)**: Es el término que hace referencia a cualquier método existente para almacenar certificados y CRLs, y así poder ser obtenidos por las entidades finales. Uno de estos métodos es el protocolo LDAP.

En ciertas jerarquías de certificación puede existir un sexto elemento que se encarga de dar información sobre la vigencia de los certificados digitales. Este elemento se llama Autoridad de Validación (*Validation Authority*, VA), y recoge la información de qué certificados han sido revocados de las CRLs.

En la arquitectura PKIX también se pueden definir unas funciones de gestión de la arquitectura, estas funciones son:

- **Registro**: Este es el proceso por el cual una entidad final (en concreto un usuario) registra sus datos directamente en una CA o por medio de una RA. También incluye procedimientos especiales de mutua autenticación, para los cuales se suelen generar claves privadas compartidas.
- **Inicialización**: Es el proceso por el cual el cliente se inicializa, de forma segura, con su clave pública y con otra información relevante de la CA en la que se ha registrado. La información de esta CA es la que se utilizará en el camino de validación de certificados. Esto es necesario para que el sistema del cliente pueda operar correctamente.
- **Certificación**: Es el proceso por el cual la CA crea un certificado que certifica que una determinada clave pública pertenece a una entidad final, y se lo retorna al sistema del cliente o lo almacena en un repositorio.
- **Recuperación del par de claves**: Este proceso permite a las entidades finales volver a obtener su par de claves, para esto las piden a una entidad autorizada de resguardo de claves. Usualmente la misma CA que emitió el certificado hace el papel de esta autoridad. Esta función es importante porque si se ha perdido el acceso a la clave de desencriptación no se podrán recuperar los datos encriptados.

- **Actualización del par de claves**: Es el proceso por el cual luego de un tiempo determinado se actualizan las claves de una entidad final y se le vuelve a emitir el respectivo certificado. Normalmente sucede cuando se cumple el tiempo de validez de un certificado o cuando el certificado ha sido revocado, y se realiza volviendo a crear un nuevo par de claves.
- **Pedido de revocación**: Ocurre cuando una persona autorizada avisa a la CA que ha ocurrido un suceso anormal y pide la revocación del certificado de una entidad final. Los motivos del pedido de revocación pueden ser el compromiso de la clave privada o el cambio de nombre de la entidad, entre otros.
- **Certificación cruzada**: Es el proceso en el cual dos CAs intercambian la información necesaria para emitir un certificado cruzado.

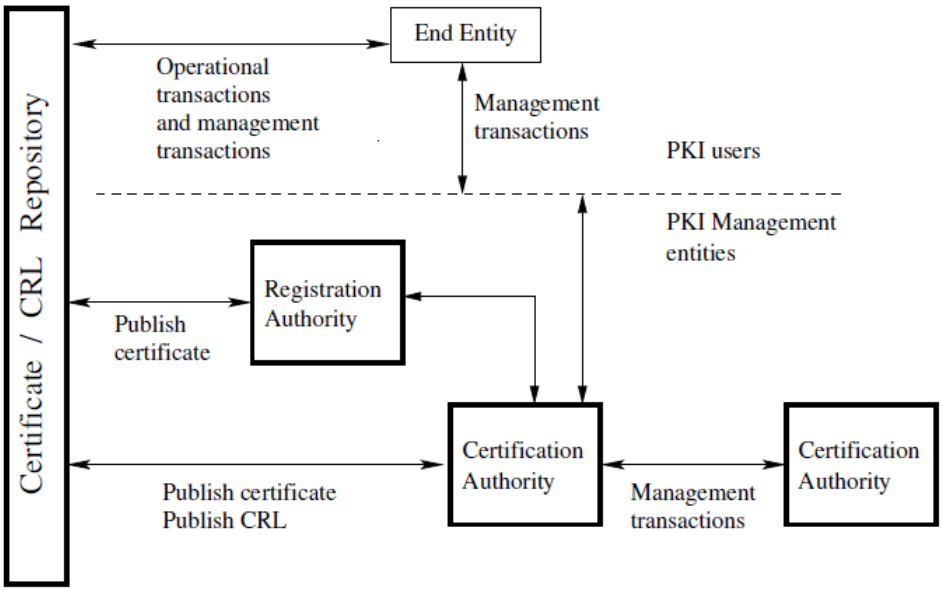


Figura 2.6. Modelo de Referencia PKIX.

2.4 Redes VANET y Modelos de Movilidad

2.4.1 Redes Ad-hoc Móviles

Una red ad-hoc se puede definir como una red compuesta por terminales fijos y móviles, o bien sólo móviles, que no dependan de una infraestructura preexistente, desplegándose de una forma espontánea en un entorno inalámbrico, sin excluir que alguno de ellos puede poseer conectividad a través de un cable. **[7]**

Una red móvil Ad-Hoc (*Mobile Ad-Hoc NETwork*) es una red inalámbrica que no requiere ningún tipo de infraestructura fija ni administración centralizada y donde las estaciones necesitan incorporar la funcionalidad de enrutamiento, retransmitiendo paquetes entre aquellas estaciones que no tienen conexión inalámbrica directa.

Tanto la estructura de la red como los nodos que la componen son altamente dinámicos y cambiantes, lo que se refleja en un enrutamiento extremadamente adaptativo. Factores a los que se debe adaptar la red son la posición de las estaciones, la potencia de la señal, el tráfico de la red, la distribución de la carga y el principal reto de las redes MANET, los cambios continuos en la topología.

Debido a que las redes MANET no requieren ningún nodo central controlando el tráfico de red, conforman un subgrupo muy significativo dentro de las redes inalámbricas. Las redes inalámbricas convencionales requieren una Estación Base (BS, Base Station), responsable de encaminar los mensajes desde y hacia los Nodos Móviles (MN, Mobile Nodes). Sin embargo, en una red Ad-Hoc, no se requiere ningún equipo extra a parte de dos o más MN trabajando cooperativamente para la red. En lugar de confiar en una BS para la transmisión de mensajes, cada MN forma una red individual y reenvía mensajes entre los distintos MN. Este comportamiento adaptativo permite a la red formarse rápidamente incluso bajo condiciones adversas.

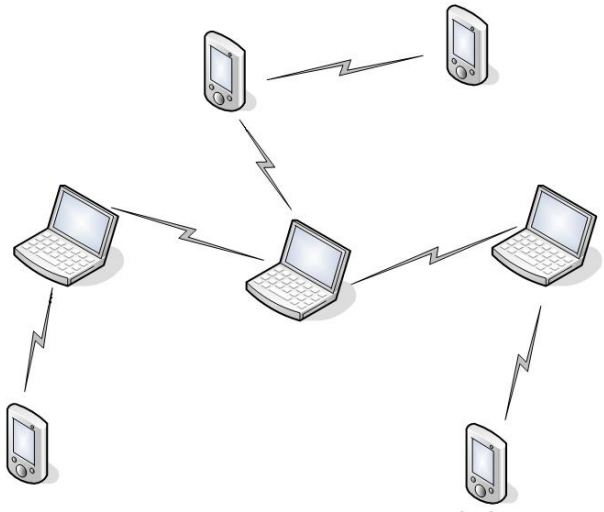


Figura 2.7. Esquema de una red MANET.

2.4.2. Características de las redes MANET

- **Topología dinámica**: Los nodos son libres para moverse libremente a diferentes velocidades, por tanto, la red debe ser capaz de cambiar aleatoriamente y de manera impredecible.
- **Multi-salto**: El camino entre origen y destino atraviesa múltiples nodos.
- **Escalabilidad**: La red Ad-Hoc puede crecer y tener varios miles de nodos (sensores, despliegue del campo de batalla, muros de vehículos...).
- **Limitaciones energéticas**: La mayoría de los nodos dentro de la red funcionan con baterías que tienen una vida limitada y por tanto la optimización de protocolos para que hagan un uso eficiente de ella se antoja crucial dentro de las redes MANET.
- **Ancho de Banda limitado**: Los enlaces inalámbricos siguen teniendo una capacidad menor que aquellos con una estructura fija.
- **Seguridad**: La capacidad de procesamiento de los nodos y la seguridad física limitada hace que este tipo de redes sean más vulnerables a posibles ataques que las redes cableadas.

2.4.3 ¿Qué es una VANET?

Una VANET o *Vehicular Ad-Hoc Network*, se trata de una red ad-hoc donde los nodos que la componen son vehículos (coches, camiones, autobuses, etc.). En este sentido, los nodos que forman la red podrían estar en movimiento (por ejemplo, circulando sobre las calles o sobre una autopista).

Por tanto, podemos decir que una VANET está compuesta por nodos que se mueven de forma arbitraria y que se comunican entre ellos (vehicle-to-vehicle), pudiendo existir también un equipo fijo próximo que formara parte de la red y que también dotará a dicha red de una conexión hacia Internet, de la misma manera que acceden los terminales móviles a Internet a través de GPRS o UMTS.

Cabe destacar que una VANET es un tipo de MANET (Mobile Ad-Hoc Network), con la diferencia que el término MANET describe sobre todo un campo de investigación académico, mientras que el término VANET está más enfocado a una de sus aplicaciones específicas: redes vehiculares. **[7]**

2.4.4 Aplicaciones de las Redes VANET

2.4.4.1. Car-to-Car Services

Este servicio permite el intercambio de información entre vehículos con la finalidad de prevenir de accidentes y reforzar la seguridad: intercambio del estado del tráfico, condiciones de la carretera, clima (posibilidades de lluvia, niebla, etc.), mantenimiento de distancias de seguridad entre vehículos, etc.

2.4.4.2. Car-to-Infrastructure Services

Ese servicio permite, además del intercambio exclusivo de datos entre vehículos, introducir una serie de estaciones base que podrían ofrecer distintos tipos de informaciones: tiempo, tráfico, etc, y adicionalmente se permitiría ampliar el radio de amplitud de la red ad-hoc formada con la posibilidad, por

ejemplo, de poder enviar a un servicio de emergencia un mensaje de auxilio, ya que cabría la posibilidad de que en esa parte de la carretera no hubiera más vehículos en ese momento.

2.4.4.3. Portal Based Services

Este sistema se basa en ofrecer a los vehículos servicios basados en infraestructura bajo una plataforma de servicios telemáticos. Dentro de estos servicios tendríamos, por ejemplo, conexión a Internet desde los vehículos, pago por tiempo conducido por parte de las aseguradoras, creación de portales Web de ciudades para consultar en tiempo real el estado de las plazas de aparcamiento, hoteles, restaurantes cercanos, etc.

2.4.5 Modelos de Movilidad

En el desarrollo de este proyecto, y a efecto de la simulación ejecutada con la aplicación implementada, hemos considerado que los clientes que harán uso del servidor desarrollado deben implementar la movilidad, es decir, la posibilidad de moverse aleatoriamente a través del escenario definido al momento de la simulación.

El escenario en sí corresponde a una gran MANET dentro de la cual existirá una CA, un Servidor de Revocaciones (que es la aplicación implementada) y un número de clientes que son los que pondrán a prueba esta aplicación, moviéndose dentro del escenario y enviando al servidor solicitudes de revocación y/o peticiones de listas de revocación de certificados (CRLs).

Es en tal sentido que el modelo de movilidad usado durante la simulación juega un rol importante al determinar el rendimiento de la aplicación y de la simulación de la MANET.

El modelo de movilidad está diseñado para describir el patrón de movimiento de los usuarios móviles, y cómo su ubicación, velocidad y aceleración cambian a través del tiempo.

Existen varios tipos de modelos de movilidad que permiten simular el comportamiento de los nodos en una MANET, los cuales están divididos en categorías de acuerdo al parámetro del cual tienen dependencia, por lo que podemos tener algoritmos dependientes del tiempo, del espacio (ubicación), de alguna restricción geográfica, o simplemente pueden ser modelos aleatorios.

En este proyecto hemos elegido dos modelos distintos para efectos del desarrollo de los clientes móviles, uno es un Modelo Aleatorio: Random Waypoint Model **[8]**[3], y el otro es un Modelo Dependiente del Tiempo: Modelo de Gauss-Markov **[8]**[4].

2.5 Redes DTN

Una Red Tolerante a Interrupciones (DTN, Disruption-Tolerant Network) representa la clase de redes donde las conexiones entre los nodos inalámbricos pueden sufrir la falta de conectividad continua.

Este tipo de redes usualmente tiene una densidad de nodos escasa, con rangos de comunicación cortos en cada nodo. La conexión a través de estos nodos puede ser interrumpida debido a los límites del alcance del radio inalámbrico, debido a la escasez de nodos móviles, recursos de energía, ataques, interferencias ambientales, o ruido **[9]**.

Este tipo de redes está caracterizado por **[10]**:

- Intermitencia en la Conectividad: Si no hay una ruta de punto a punto entre el origen y el destino, la comunicación punto a punto usando protocolos TCP/IP no funciona. Se requerirán otros protocolos.
- Retardo grande o variable: Adicionalmente a la intermitencia, largos retardos de propagación entre los nodos y retardos de espera variables, añadidos al retardo de la ruta de punto a punto pueden vencer a los protocolos de Internet y a las aplicaciones que requieran de acuses de recibo inmediatos o retorno de información.

[3] **Ref. 7, Capítulo 2. Random Based Mobility Models: The random-waypoint model p. 4**
[4] **Ref. 7, Capítulo 3: Mobility models with temporary dependency: Gauss-Markov Mobility model. p. 13**

- Tasas de datos asimétricas: Internet soporta asimetrías moderadas en tasa de datos bidireccionales (como en la TV por cable o ADSL). Pero si estas asimetrías son muy grandes, los protocolos de conversación son derrotados.

- Elevadas tasas de error: Errores en el envío de bits requieren corrección (lo que requiere más bits y mayor procesamiento) o retransmisión del paquete entero (lo que resulta en mayor tráfico de red).

Las tecnologías DTN pueden ser diversas, incluyendo no solo radio frecuencias, sino también la banda UWB (ultra-wide band), el espacio libre óptico, y tecnologías acústicas (sonar o ultrasonido).

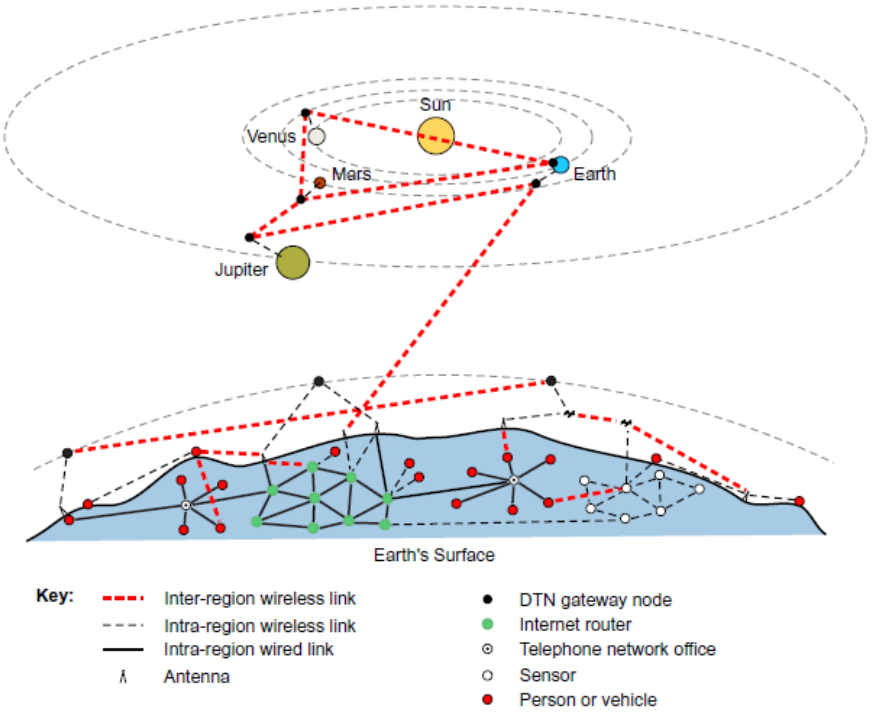


Figura 2.8. Esquema de una red DTN.

2.6 Notación de Sintaxis Abstracta 1 (ASN.1)

ASN.1 (Abstract Syntax Notation 1) es un lenguaje que permite definir estándares independientemente de la implementación. Es el lenguaje que emplean los autores de estándares. Esta norma permite representar datos independientemente de la máquina que se esté usando y sus formas de representación internas **[11]**.

ASN.1 no tiene relación con ningún estándar, método de codificación, lenguaje de programación o plataforma de hardware en concreto. Se trata simplemente de un lenguaje para definir estándares. Dicho de otro modo, los estándares se escriben en ASN.1.

El formato ASN.1 define:

- *Qué es un "tipo".*
- *Qué es un "módulo" y qué apariencia debe tener.*
- *Qué es un "entero".*
- *Qué es un valor "booleano".*
- *Qué es un "tipo estructurado".*
- *Qué significan ciertas palabras clave (por ejemplo, BEGIN, END, IMPORT, EXPORT, EXTERNAL, etc.).*
- *Cómo "etiquetar" un tipo para que se pue*da codificar correctamente.

La sintaxis de transferencia especifica cómo se codifican los distintos tipos de datos. Define la forma de codificar en el transmisor y decodificar en el receptor los valores expresados con ASN.1.

Este estándar no define cómo se han de codificar esos datos, sino que es una sintaxis abstracta para indicar el significado de los datos. Para la codificación de los datos se usan otras normas como: BER (Basic Encoding Rules) (BER - X.209), CER (Canonical Encoding Rules), DER (Distinguished Encoding Rules), PER (Packed Encoding Rules) y XER (XML Encoding Rules). **[12]**

La sintaxis BER, junto con dos subconjuntos de BER: Canonical Encoding Rules (CER) y Distinguished Encoding Rules (DER), están definidas por el documento de estándares X.690 de la UIT-T, el cual es parte de las series de documentos ASN.1. Cada elemento de datos está codificado usando la codificación tipo-longitud-valor, es decir, por un identificador de tipos, una descripción longitud, los elementos de datos actuales, y donde sea necesario, un marcador de fin-de-contenido. **[13]**

Por ejemplo, cuando codificamos un valor construido (esto es, un valor que está compuesto de múltiples valores ya codificados más pequeños), el emisor puede usar una de las tres formas diferentes para especificar la longitud de los datos. Estas implementaciones confían en la flexibilidad que BER suministra para usar lógica de codificación que es más fácil de implementar, pero redunda en una corriente de datos mayor de lo necesario.

Los certificados digitales X.509, así como las listas de revocación (CRLs), al ser estándares definidos, son representados en notación ASN.1. Para el desarrollo del proyecto, en el momento de leer los certificados, y también al momento de escribir las CRLs, se utiliza siempre la notación ASN.1 y la codificación DER.

Capítulo 3

LA REVOCACIÓN DE CERTIFICADOS

Como vimos en el capítulo anterior, cuando Alice quería comunicarse de manera segura con Bob, la presentación por parte de Bob de su certificado no era evidencia suficiente para Alice. Alice también requería “validar” dicho certificado. La validación de un certificado comprende 2 mecanismos:

- Validación de la ruta de certificación: Este mecanismo es necesario únicamente cuando Alice quiere establecer una transacción segura con Bob y la CA que ha firmado el certificado de Bob es diferente a la CA que ha firmado el certificado de Alice. En este caso, Alice necesita un mecanismo para crear y validar una cadena de certificados que finalmente certifiquen la CA de Bob.

- Verificación de Estado del Certificado: Este mecanismo define cómo se comunicará el estado de un certificado (revocado, no revocado, u otro tipo de estado) a los otros usuarios interesados. Entre otras cosas, la verificación del estado debe definir el formato de datos que será empleado durante el intercambio de datos, y las entidades que forman parte de este proceso.

En este capítulo veremos a fondo dos conceptos importantes que nos permitirán entender el funcionamiento de la aplicación desarrollada durante este proyecto, que son:

- La Revocación de Certificados,
- La definición de un Mecanismo basado en el usuario para emitir la información del estado de un certificado (revocado o no revocado) en redes tolerantes a interrupciones: el Riesgo.

3.1 Revocación de Certificados

La Revocación de un Certificado es un procedimiento que invalida un certificado determinado como credencial de seguridad de confianza antes de la caducidad natural de su periodo de validez.

Una PKI depende de la comprobación distribuida de las credenciales en las que no es necesario que haya comunicación directa con la entidad de confianza central que acredita dichas credenciales. Esto crea la necesidad de distribuir información de revocación de certificados a las personas, equipos y aplicaciones que intentan comprobar la validez de los certificados. La necesidad de datos de revocación y su caducidad varían, según la aplicación y la implementación de comprobaciones de revocación de certificados.

Para admitir de manera eficaz la revocación de certificados, el cliente deberá determinar si el certificado es válido o si ha sido revocado. Existen varios métodos para conocer el estado de un certificado, de los cuales el que será utilizado en este proyecto para la implementación del Servidor de Revocaciones es la publicación de listas de revocación de certificados (CRL, Certificate Revocation Lists).

Estas listas de revocaciones de certificados son listas completas, y firmadas digitalmente, de certificados no caducados que han sido revocados. Los clientes pueden recuperar esta lista, y posteriormente, almacenarla en caché (dependiendo del ciclo de vida configurado en la CRL) y utilizarla para verificar certificados que se presentan para su uso.

Como las CRLs pueden llegar a ser grandes, dependiendo del tamaño de la entidad Emisora de Certificados, también pueden publicarse listas de

revocaciones de certificados de diferencias (Delta-CRL). Estas contienen solo los certificados revocados desde que se publicó la última lista de revocaciones de certificados base.

3.1.1 Entendiendo la Revocación

El problema de la revocación es aún mucho más complicado de lo que parece a primera vista. Para entender el proceso de Revocación, detallaremos el siguiente ejemplo donde tenemos un escenario donde se requiere de la revocación de un certificado:

Consideremos un certificado digital c_1 emitido para Alice, el cual contiene la clave pública de Alice (KU_{Alice}) y la firma de su CA_1. Supongamos que Alice tiene otro certificado digital c_2, emitido por otra autoridad CA_2, diferente a CA_1.

El problema se presenta si CA_1 revoca el certificado c_1. Esto significa que c_1 ya no es válido, pero no nos dice nada acerca de c_2, el cual contiene la misma identidad y la misma clave pública que el certificado c_1. Revocar c_1 puede implicar deshacer alguno de los siguientes elementos:

1. KU_{Alice}: lo cual significa que la clave pública de Alice yo no es confiable debido a que puede haber sido comprometida.
2. $KU_{Alice} \leftrightarrow DN_{Alice}$: La asociación entre la identidad (DN) de Alice y su clave pública puede haber sido comprometida.
3. CA_1 *asociando* $KU_{Alice} \leftrightarrow DN_{Alice}$: el emisor ya no puede garantizar la asociación entre las claves y sus respectivas identidades.

Cada uno de estos casos implica algo diferente, y las acciones a llevar a cabo para efectuar la revocación también son distintas en todos los casos. Por ejemplo, si consideramos el primer caso donde la revocación de c_1 denota que la clave pública del sujeto ha sido comprometida. En este caso, el hecho de que c_1 haya sido revocado debe causar que todos los demás certificados que involucren la clave pública del sujeto KU_{Alice} ya no sean válidos. Por tanto, no solo c_1 pasará a ser inválido, sino que c_2 ahora también sería inválido. Idealmente, si algún certificado conteniendo la clave pública de un sujeto es revocado por razones de compromiso de clave, todos los certificados

relacionados a este quedarían inmediatamente revocados. Sin embargo, es obvio que es muy difícil garantizar este comportamiento.

En el segundo caso, tenemos una situación un poco más confusa. Como los dos certificados, c_1 y c_2 pertenecen a Alice, las partes implicadas (las CAs) pueden elegir razonablemente rechazar c_2 si saben que c_1 está revocado (aunque c_2 no haya sido revocado explícitamente).

Finalmente, en el tercer caso tendremos la revocación del certificado c_1 debido a que el emisor de ese certificado ya no tiene ninguna relación con la clave pública del sujeto. Esta revocación de c_1 no debe tener ningún impacto sobre el certificado c_2, debido a que no hay ninguna notificación que implique que la seguridad de la clave pública del sujeto KU_{Alice} haya sido comprometida.

En el presente proyecto, para el proceso de revocación, asumimos que todos los certificados a revocar son independientes entre sí, por lo que no hace falta una comprobación exhaustiva del impacto de la revocación de un certificado con respecto a otros emitidos por la misma CA.

3.1.2 Razones de Revocación

El hecho de que el acto de revocar un certificado en particular necesita ser clasificado de alguna manera es reconocido por los autores del estándar X.509. En X.509, es posible incluir un código de razón por cada certificado revocado. Estos códigos de razón de revocación pueden especificar varias situaciones, entre ellas:

- Compromiso de clave.
- Compromiso de la CA.
- Cambio de Afiliación de CA (incluyendo también cambios en el nombre del sujeto).
- Derogado (superseded): el certificado se ha reemplazado debido a que existe uno más actualizado.
- Cese de operación (el certificado ya no es necesitado para su propósito original).

3.1.3 El paradigma de la Revocación de Certificados

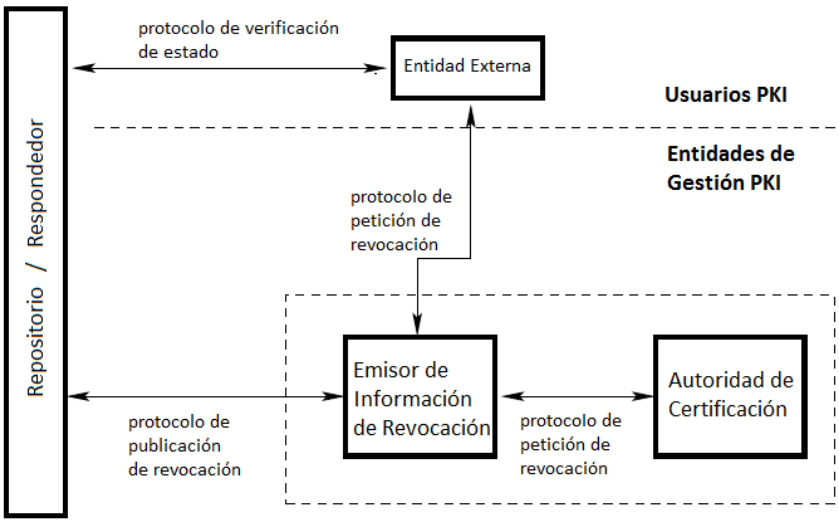


Figura 3.1. El proceso de Revocación

La figura 3.1 presenta el modelo de referencia que se utiliza en este proyecto para describir el paradigma de la revocación de certificados.

El proceso de revocación empieza con una petición de revocación de un cierto certificado. Usualmente la revocación es solicitada por el dueño del certificado, por una autoridad representativa o por la CA emisora. Aunque en general, la Política de Certificación (CP, Certification Policy) es la que define quién es capaz de solicitar una revocación en el dominio de la PKI.

Para revocar el certificado, la entidad autorizada genera la solicitud de revocación y la envía al Emisor de Datos de Revocación (RDI, Revocation Data Issuer). El término *RDI* lo usamos para definir a la TTP que tiene la Base de Datos principal de certificados revocados.

La RDI también es responsable de transformar los registros de revocación de la Base de Datos en “Información de estado”. Esta información de estado tiene el formato adecuado que le permitirá ser distribuida a las entidades externas.

La información de estado debe incluir los siguientes campos:

- **Certificate Issuer (Emisor del certificado):** Es el DN de la CA que emitió el certificado revocado.
- **Validity Period (Tiempo de validez):** Tiempo de vida de la información de estado. Siempre es menor que el tiempo de vida del certificado.
- **Issuer Name (Nombre del Emisor):** es el nombre de la TTP que emitió la información de estado.
- **Cryptography evidence (Evidencia criptografía):** Debe demostrar que la información de estado fue emitida por la TTP.
- **Serial number (número de serie):** Número de serie del certificado revocado.
- **Revocation Date (Fecha de Revocación)**
- **Reason Code (Código de Razón de Revocación):** Usualmente se utilizan los siguientes códigos: *unspecified, keyCompromise, cACompromise, affiliationChanged, superseded, removeFromCRL, cesationOfOperation y certificateHold.*

En la mayoría de los Sistemas de Revocación, las Entidades Externas no tiene conexión directa con la RDI. En vez de esto, la RDI publica la Información de Estado en “repositorios” o “entidades de respuesta”. La principal función de ambos es la de responder las peticiones concernientes al estado de los certificados (Verificación de Estado).

La diferencia entre estos es que los repositorios no son TTPs, esto es, solamente almacenan la Información de Estado computada por la RDI; mientras que las entidades de respuesta son TTPs que tienen una clave privada y que proveen una firma digital en cada respuesta.

3.1.4 Protocolos de Gestión de Certificados

El protocolo CMP (Certificate Management Protocol) es el que define el formato de los paquetes para las operaciones asociadas a la creación y administración de certificados.

Estas operaciones incluyen: Establecimiento de la CA, Inicialización de Entidades, Registro, Creación de certificados, Actualización del par de claves, Actualización de certificado, Operaciones de recuperación de clave, Publicación de certificado, y finalmente **Publicación de listas de revocación** y **Peticiones de Revocación.**

Únicamente estas dos últimas operaciones son relevantes para este proyecto. La plataforma implementada permite recibir y procesar Solicitudes de Revocación de Certificados a través del protocolo SCRP (Simple Certificate Revocation Protocol), el cual es una simplificación del protocolo CMP y que contiene únicamente los datos que consideramos relevantes para la solicitud de la revocación. El protocolo SCRP fue desarrollado por el director de este proyecto para la implementación de la aplicación Cervantes, sobre la cual está basado el funcionamiento de esta nueva plataforma desarrollada.

3.1.5 Protocolos de Verificación de Estado

Los protocolos de verificación de estado nos permiten conocer la Información de Estado de un certificado. A efectos de este proyecto, nos remitiremos a que estos protocolos nos permiten conocer si un certificado está revocado o no.

A continuación, presentamos una clasificación de los protocolos más utilizados para la Verificación de Estados de Certificados, haciendo mayor incidencia en el método de Publicación de Listas de Revocación de Certificados, ya que este método es el que se usa en la plataforma implementada.

3.1.5.1 Protocolos Basados en Listas

CRL Tradicional (RFC 2580 [1])

El planteamiento más simple para la Verificación de estados es sin dudas el uso de CRLs. Las listas de revocación han sido parte del estándar X.509 desde su primera versión.

Una CRL es una "lista negra" que contiene todos los certificados que se encuentran revocados, y que aún no han expirado, emitidos por una cierta CA. La integridad y autenticidad de una CRL es provista por una firma digital añadida a la CRL. En el caso de las CRL Tradicionales, la entidad autorizada para emitir esta firma es la CA que firmó el certificado emitido. Debido a que tienen una firma digital, las CRLs pueden ser distribuidas a través de repositorios públicos sin comprometer la seguridad de las mismas.

Actualmente existen dos versiones de CRL en el estándar X.509. En el desarrollo de este proyecto se ha utilizado la X.509v2 CRL, que ha sido aprobada por la IETF para permitir la interoperabilidad en Internet.

La estructura genérica del estándar X.509v2 CRL la podemos apreciar en la siguiente figura:

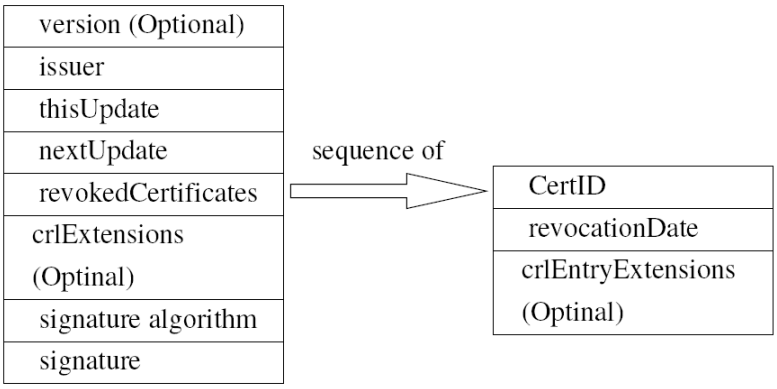


Figura 3.2. Modelo de Referencia de una CRL X.509 v2

- **Version (Versión):** Indica simplemente la versión de la CRL.
- **ThisUpdate (Fecha actualización):** Fecha de creación de la CRL.
- **NextUpdate (Próxima actualización):** Opcionalmente, se puede especificar la fecha en la que está previsto emitir la siguiente CRL.
- **RevokedCertificates (Certificados revocados):** Una lista, en la cual hay un registro por cada certificado revocado. Cada registro contiene:
    - RevocationDate (Fecha de Revocación).
    - entryCRLExtentions: Extensiones opcionales por cada registro. Las extensiones más relevantes son:
        - *Reason Code:* Razón por la que se ha revocado el certificado.
        - *CertificateIssuer:* El nombre del emisor del certificado
- **CRLExtensions (Extensiones de CRL):** Campo opcional que puede incluir extensiones con información adicional. Las más relevantes son:
    - **CRLNumber:** Número de serie único, relativo al emisor, para esta CRL. Este número permite la detección de CRLs faltantes para cualquier emisor de CRLs.
    - **IssuingDistributionPoint:** Indica el nombre del punto de distribución de CRL y el tipo de certificados contenidos dentro de la CRL.
    - **DeltaCRLIndicator:** Este campo indica si la CRL es una Delta-CRL o si es una CRL Tradicional.
- **Signature (Firma)**
- **Signature Algorithm (Algoritmo de Firma)**

CRL Sobre-emitida (Overissued CRL)

El método de CRLs tradicionales implica a una CA publicar periódicamente CRLs. Dado que las CRLs pueden tener un gran tamaño, normalmente son guardadas en caché por el cliente durante el periodo de validez para mejorar su funcionamiento. El cacheo de CRLs facilita también la habilidad de verificar certificados mientras trabajamos sin conexión.

El problema con el método tradicional de emitir una CRL es que todas las CRLs almacenadas en caché por distintos usuarios expiran al mismo tiempo. Inmediatamente luego de que la CRL expira, y una nueva CRL es emitida, todos los usuarios requerirán obtener esta nueva CRL. Como resultado, habrá tasas de solicitud de CRLs muy elevadas cuando se emita una nueva, y en cambio las solicitudes serán casi inexistentes durante el periodo en que la CRL es aún válida.

Para solucionar esto, existe el método de las CRL sobreemitidas. Este método consiste simplemente en emitir más de una CRL durante un periodo de validez. El resultado será que cada uno de los usuarios tendrá almacenada en caché una copia de la CRL, la cual expirará en tiempos distintos. De esta manera, los picos de solicitudes de descarga de CRL procesadas por el servidor quedarán más repartidos a lo largo de un periodo de validez.

3.1.5.2 Protocolos Basados en Firmas en línea

El otro mecanismo popular en la verificación de estado de un certificado es aquel que se basa en emisión de firmas en línea. El protocolo más conocido es el OCSP (Online Certificate Status Protocol, RFC 2560 **[14]**).

OCSP es un protocolo relativamente simple de peticiones y respuestas, que permite que las aplicaciones que emplean los certificados determinen el estado de revocación de un certificado determinado. Los datos del estado están disponibles en línea desde una entidad denominada Módulo de Respuestas OCSP.

Una petición OCSP está compuesta por la versión del protocolo, el tipo de petición, y uno o más identificadores de certificado. Este identificador consiste en un hash del DN del emisor del certificado, el hash de la clave pública del emisor y del número de serie del certificado. Adicionalmente se pueden añadir extensiones.

La respuesta también es bastante sencilla, consiste de un identificador del certificado, el estado del certificado: good (bueno), revoked (revocado), o unknown (desconocido), y el intervalo de validez de la respuesta asociada a cada identificador de certificado especificado dentro de la petición original. Si el

estado es “revocado”, se añaden también la fecha de revocación, y opcionalmente, la razón de la revocación.

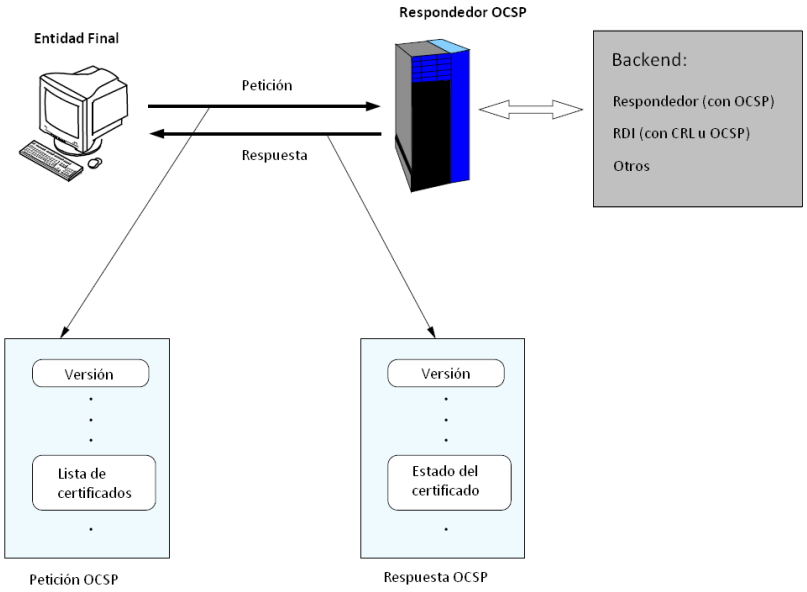


Figura 3.3. Uso de OCSP.

3.1.5.3. Otros métodos

Existen otros métodos para la verificación de estados, que al igual que OCSP, quedan fuera del alcance de este proyecto, y no son tomados en cuenta en el desarrollo de la aplicación implementada. Sin embargo, presentamos como referencia una lista con estos métodos adicionales:

- Métodos basados en Árboles de Merkle (*MHT*, *Merkle Hash Tree*):
- Listas de emisión con hashing conmutativo (*SLCH*, *Skip Lists with Commutative Hashing*).
- Sistema de Revocación de Certificados (*CRS*, *Certificate Revocation System*).
- CRS Jerárquico (*H-CRS*).

3.2 Riesgo

3.2.1 Introducción

En una red DTN, los usuarios requieren soluciones para controlar la posible ausencia de conectividad con las autoridades verificadoras ubicadas en el lado de la infraestructura de la red. Adicionalmente, la verificación de un certificado requiere validarlo en tiempo real, sin embargo, el problema en una DTN es que los usuarios pueden estar desconectados de la fuente de Información de Estado de Certificado (CSI, *Certificate Status Information*).

Por tanto, se propone una forma novedosa de emitir CRLs de manera que los usuarios puedan tomar mejores decisiones acerca de la validez de un certificado durante un intervalo de desconexión. En vez de emitir CRLs periódicamente, la CSI es emitida de acuerdo a un nuevo criterio basado en el usuario. Este novedoso mecanismo basado en el usuario hará que los intervalos de emisión de CSI sean variables y dependientes del riesgo de la Información de Estado almacenada en caché. **[15]**

3.2.2 El problema de la Verificación de CSIs en redes DTN

En general, los usuarios requieren del uso de CSI almacenadas en caché para validar certificados. Sin embargo, como los mecanismos basados en infraestructura usualmente emiten CSI periódicamente, esta CSI cacheada no podría ser siempre la más reciente que pueda existir.

Los usuarios tienen dos mecanismos disponibles para asegurar que una CSI está actualizada:

- El primer mecanismo, y el más sencillo está basado en el tiempo. El usuario puede mirar el campo *thisUpdate* de la CSI, y dependiendo de cuan antiguo sea este parámetro, tomar una decisión considerando o no la CSI desactualizada.
- El segundo mecanismo implica el uso de un valor de seguridad (*nonce*) para asociar la CSI de respuesta a una determinada solicitud de CSI.

Este valor "*nonce*" es un número pseudo-aleatorio emitido por el cliente para asegurar que una respuesta antigua no pueda ser reusada.

En una red DTN, la principal desventaja de la evaluación de CSI basadas en el tiempo es que puede ocurrir que los usuarios no estén conectados cuando se está emitiendo una CSI nueva. De tal manera, hasta que el usuario no sea capaz de obtener una versión actualizada de la CSI, quedará inadvertido del proceso de revocación que se lleve a cabo en la red. De la misma manera, el segundo mecanismo tiene el mismo problema. Por tanto, se puede concluir que ninguno de los mecanismos previos se adapta a las redes DTN.

En la siguiente figura, podemos apreciar cuan crítico puede ser el proceso de revocación en una DTN.

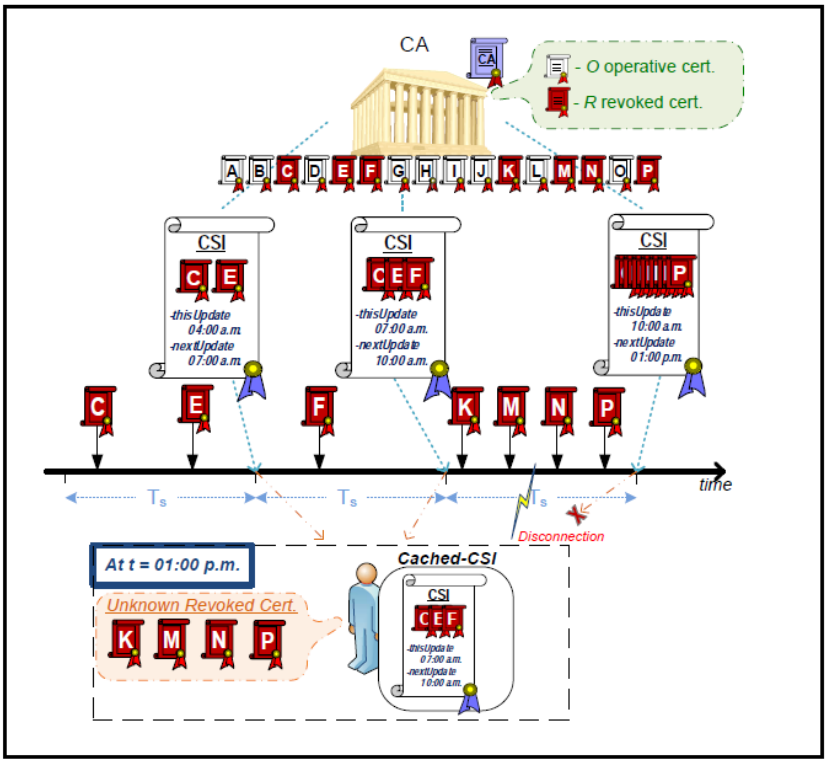


Figura 3.4. Ejemplo de emisión periódica de CSI.

En este ejemplo, después del primer intervalo de emisión T_S, el usuario sabe que los certificados C y E han sido revocados, y que la siguiente actualización de la CSI será a las 7.00am. De la misma manera, después de otro intervalo

T_S, el usuario sabe que los certificados C, E y F está revocados y que la siguiente actualización de CSI será a las 10.00am. Probablemente, el usuario no considerará este tiempo T_S = 3 horas como un intervalo grande si solamente se revoca un certificado cada 2 horas, pero esta impresión podría cambiar si la tasa de revocación creciera hasta tener algunos cientos de certificados revocados por día.

El problema es más grave si consideramos la naturaleza de las redes DTN: si consideramos que en algún momento antes de las 10.00am el usuario pierde la conexión con la PKI, de manera que no puede descargar la actualización de la CSI. En ese caso, si quisiera operar con un certificado después de las 7.00am, no estaría seguro de cuál es su estado. Si a las 8.00am quisiera usar un certificado que no está en la CSI almacenada en caché, podría pensar que al solo haber pasado una hora de la emisión de la última CSI, la probabilidad de que este certificado este revocado es muy baja. Sin embargo, el usuario no tiene suficiente información para tomar esta decisión. Podríamos tener el mismo caso anterior, donde la tasa de revocaciones se haya incrementado, con lo que esa probabilidad de que el certificado esté revocado, en ese caso, sería alta.

Por tanto, este nuevo mecanismo del riesgo propone cambiar el punto de vista de los mecanismos basados en infraestructura para emitir una CSI, debido a que de esta manera no se provee al usuario de suficiente información para tomar una buena decisión durante los periodos de desconexión.

3.2.3 Planteamientos preliminares

Como primer planteamiento tenemos un mecanismo basado en el usuario que pueda incluir parámetros adicionales dentro de la CSI. Por ejemplo, se podría incluir el número total de certificados y el número total de certificados revocados dentro de extensiones, por ejemplo, en una CRL extendida. La principal desventaja de este mecanismo es sin duda que modifica la estructura estándar de la CSI, por lo que la compatibilidad entre usuarios no se podría garantizar.

En el segundo mecanismo basado en usuario, el lado de la infraestructura podría emitir una CSI estándar, pero usando diferentes intervalos de tiempo

durante el proceso de revocación. El lado de la infraestructura adaptaría los intervalos de emisión de acuerdo a las condiciones del proceso de revocación, y el usuario podría inferir alguna información adicional de estos intervalos. El usuario miraría el tiempo transcurrido desde la emisión de la CSI (*thisUpdate*) y la emisión de la siguiente CSI (*nextUpdate*).

En la siguiente figura se presenta un ejemplo de este último mecanismo. Este ejemplo considera que se define un umbral R_{th}, el cual podría definir que para un número de certificados inferior a este umbral, el intervalo de emisión T_S de la CSI se duplicaría, y para valores mayores a este umbral, el intervalo de emisión se reduciría a la mitad.

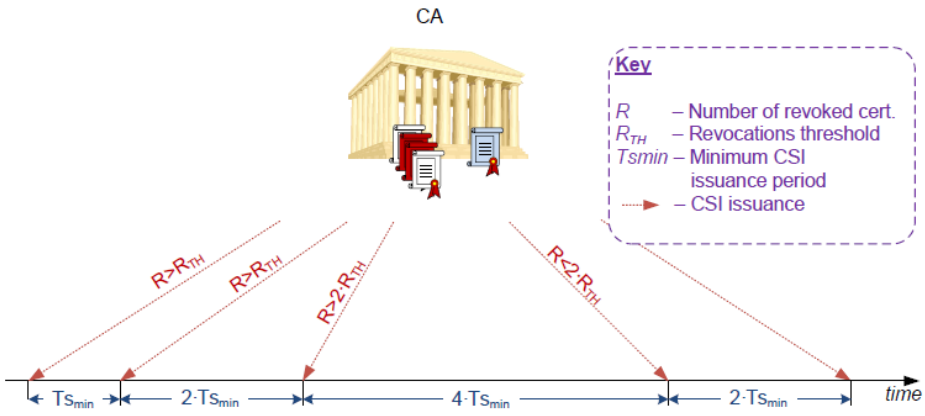


Figura 3.5. Ejemplo de un mecanismo basado en usuario.

3.2.4 Mecanismo basado en el usuario para emitir una CSI

Si tomamos como base el mecanismo descrito anteriormente, la mejor forma de proveer información al usuario sobre el proceso de revocación es codificándola en la longitud del intervalo de emisión de CSI. De esta manera, se define el cálculo del riesgo como un nuevo criterio basado en un análisis estadístico del proceso de revocación. Con este criterio se podría determinar la duración del intervalo de manera que el usuario pueda inferir información útil acerca del proceso de revocación, y poder tomar una buena decisión.

3.2.4.1 Certificados desde la perspectiva de una PKI

El conjunto de certificados existentes, desde la perspectiva de una Infraestructura, es muy sencillo de definir. Consideremos ***C*** como el conjunto de **certificados no expirados,** T_C como el intervalo de validez de un certificado (antes de que expire), y λ_C como una tasa (constante) de emisión de certificados.

Es fácil determinar que el valor esperado de ***C*** viene dado de acuerdo a:

$$E[C(t)] = C = \lambda_C T_C$$

Del mismo modo, definimos ***V*** como el número de **certificados revocados no expirados.** Este conjunto V será un subconjunto de C, ya que los certificados revocados no expirados están incluidos dentro del conjunto de todos los certificados emitidos no expirados. *V(t)* será un proceso estocástico modelado como una fracción de los certifcados no expirados *C(t):*

$$V(t) = p(t)C(t), p(t) \leq 1$$

Como los procesos *p(t) y C(t)* son independientes, el valor esperado de ***V*** puede expresarse como:

$$E[V(t)] = V = E[p(t)]E[C(t)] = pC$$

Todos los certificados no expirados restantes que no han sido revocados conforman el conjunto ***G:*** son los certificados buenos. Y es fácil concluir que:

$$G(t) = \big(1 - p(t)\big)C(t)$$

$$G = (1 - p)C$$

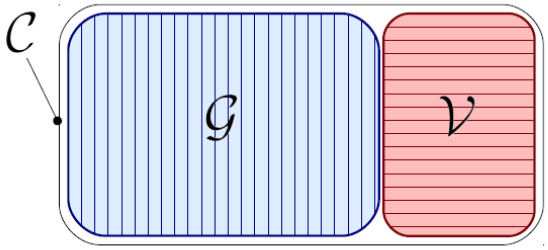


Figura 3.6. Conjuntos de certificados desde la perspectiva de la PKI.

3.2.4.2 Certificados desde la perspectiva del usuario

A pesar de que la CA conoce exactamente cuando un certificado está revocado, es imposible comunicar este cambio de estado a un usuario dentro de la DTN debido a los intervalos de desconexión. Entonces, el usuario podría considerar un certificado como operativo cuando en verdad la CA sabe que ese certificado está realmente revocado.

Definimos los siguientes conjuntos:

O, que viene a ser el conjunto de **certificados operativos,** esto es, el grupo de certificados que no están revocados de acuerdo a la última CSI obtenida por el usuario y que no hayan expirado en un instante de tiempo t.

U, es el conjunto de **certificados operativos revocados desconocidos**, que es el conjunto de certificados operativos *O* que están revocados en un instante de tiempo t. *Estos han sido revocados, pero el usuario no se ha enterado de esto.*

La CA puede calcular el número de certificados operativos revocados desconocidos como la diferencia entre los certificados operativos y el conjunto de **certificados buenos, *G.***

$$U = O - G$$

Por otro lado, el conjunto ***K*** determina el conjunto de **certificados operativos revocados conocidos**, que consiste en la lista de certificados que se encuentran dentro de la CSI almacenada en caché que aún no hayan expirado.

Podemos analizar la evolución de estos conjuntos de la siguiente manera:

En el instante t_0 = *thisUpdate*, el usuario tiene la información más actualizada de la CSI. En este momento los certificados se pueden dividir en dos el conjunto de certificados no expirados: certificados revocados (*V)* y certificados operativos (*O*). En este instante, el usuario conoce el estado de todos los certificados sin error alguno.

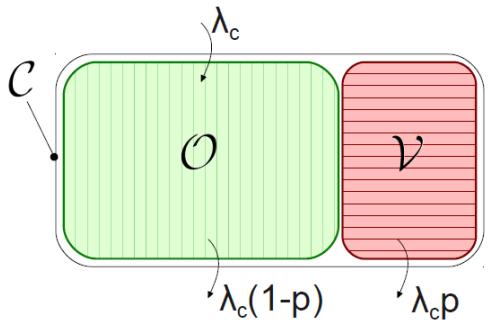


Figura 3.7. Conjunto de certificados desde la perspectiva de usuario para el instante t_0.

Consideramos que el usuario ya no es capaz de conectarse más a la infraestructura. A medida que el tiempo pasa, el conjunto de certificados operativos *O* se incrementará debido a los certificados revocados desconocidos *U*, y el usuario tendrá que tomar decisiones acerca de usar un certificado operativo asumiendo un cierto riesgo.

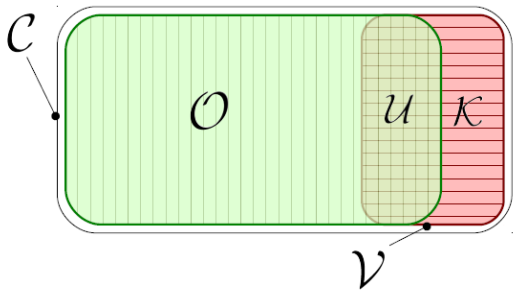


Figura 3.8. Conjunto de certificados desde la perspectiva del usuario para un instante $t = t_0 + \Delta t$

En el instante $t_0 + T_C$, todos los certificados que se encontraban en la CSI habrán expirado. Esto significa que todos los certificados no expirados estarán operativos, y cualquier certificado revocado será desconocido para el usuario.

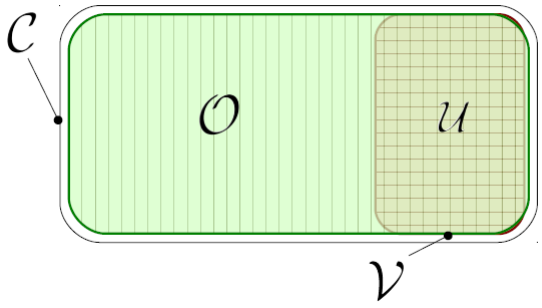


Figura 3.9. Conjunto de certificados desde el punto de vista del usuario para un instante $t_o + T_C$.

3.2.5 Cálculo de la función del riesgo

Para realizar este cálculo, se define un criterio basado en el análisis estadístico de los conjuntos antes estudiados. Con este criterio, la CA será capaz de variar la longitud de los intervalos de emisión para codificar en esos intervalos información extra acerca del proceso de revocación, de manera que estén percatados del riesgo que estarían tomando al usar un certificado operativo que no esté necesariamente dentro del conjunto de certificados buenos.

Podemos definir el riesgo de la siguiente manera:

El riesgo r(t) es la probabilidad de considerar un certificado como válido cuando su estado real (conocido por la PKI) es revocado en un instante de tiempo t. **[15]**

Esto es, podemos calcular el riesgo como la relación entre los certificados revocados desconocidos y los certificados operativos:

$$r(t) = \frac{E[U(t)]}{E[O(t)]}$$

$$r(t) = \frac{p(t - t_0)}{(1-p)t_0 + p(t - t_0)}$$

El cálculo del riesgo está fuera del alcance de este proyecto. Para detalles sobre el proceso matemático del cálculo del riesgo, consultar la referencia **[15]**, en el *Capítulo V: A User-Based Mechanism for issuing CSI,* págs. 6 - 10.

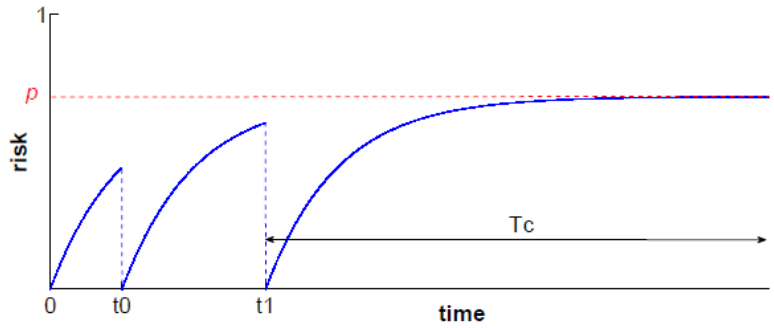


Figura 3.10. Función del riesgo.

Como alternativa se define un mecanismo, que es el utilizado en este proyecto, de Emisión con Riesgo Constante (CRI, Constant Risk Issuance), el cual permite que la CA pueda mantener la emisión de la CSI por debajo de un riesgo determinado. Este mecanismo se basa en la elección de un umbral de riesgo (r_{th}). Este umbral es publicado por la PKI a los usuarios, de manera que estos puedan calcular a partir de este valor de *p* a partir del umbral:

$$p = \frac{r_{th} T_C}{(1 - r_{th} T_S) + r_{th} T_C}$$

Capítulo 4

DESARROLLO DE LA PLATAFORMA

4.1 Descripción de la Plataforma

En este capítulo haremos una descripción de la plataforma implementada durante el desarrollo de este proyecto. Esta aplicación consiste en una plataforma cliente/servidor, desarrollada por el autor de este proyecto.

Podemos dividir el desarrollo de la plataforma en dos partes:

El servidor, el cual permite gestionar el proceso de Revocación de Certificados Digitales, esto es, se encarga de escuchar las solicitudes de revocación enviadas por diversos clientes, procesarlas, y enviar la respuesta correspondiente a cada cliente. Además, también permite la Verificación del Estado de un Certificado, de manera que mantiene actualizada una lista de los certificados revocados, generada a partir de la Base de Datos que contiene un registro por cada Certificado Digital revocado por el Servidor. Para cumplir

estas tareas, el servidor está preparado para procesar dos tipos de solicitudes: Solicitudes de Revocación de Certificados y Solicitudes de Descargas de CRL.

Por último, la aplicación también implementa la evaluación de un valor denominado *riesgo*, el cual es la base de un mecanismo basado en el usuario que permite que los usuarios dispongan de más información acerca del proceso de revocación llevado a cabo en el servidor, de manera que durante momentos de desconexión, puedan tomar las mejores decisiones posibles sobre usar un certificado del que no se está seguro su estado, con el menor riesgo posible a que dicho certificado este realmente revocado.

En el lado del ***cliente***, se ha implementado una sencilla interfaz gráfica para poder simular las solicitudes de los clientes: un cliente de Test para peticiones SCRP de Revocación de Certificados, y un cliente de Test para la solicitud de descarga de una CRL.

4.2 Herramientas Informáticas empleadas durante la implementación

- Lenguaje de Programación: Python 2.6.5
- Base de Datos: MySQL 5.1
- Herramientas de Desarrollo:
  - IDLE / Notepad++ (Windows)
  - PyShell / PyCrust (Linux)

Para este desarrollo se ha decidido utilizar Python debido a que es una herramienta de programación multiplataforma, tiene una sintaxis de programación bastante sencilla e intuitiva, y además porque existen diversos módulos y librerías que permiten un fácil manejo de herramientas de seguridad y de criptografía.

Durante el desarrollo, la principal desventaja que se encontró fue encontrar una librería que tuviera soporte para la estructura de datos ASN.1 en Python.

Como en Python no existe una librería que permita compilar y crear las clases Python a partir de la descripción ASN.1 de un elemento, se utiliza una librería que permite utilizar los elementos básicos de la notación ASN.1, y a partir de ella se ha parseado manualmente cada uno de los componentes de la estructura ASN.1 de algunos elementos a utilizar: CRLs, solicitudes SCRP y respuestas SCRP. En el caso de los certificados X.509, se utiliza la librería M2Crypto que ofrece una clase con soporte para certificados X.509.

4.2.1 Librerías y módulos de Python utilizados durante el desarrollo

pyasn1

Esta librería es una implementación de los tipos y códecs ASN.1 para el lenguaje de programación Python. Inicialmente fue escrita para soportar el protocolo SNMP, pero luego fue generalizada para ser adecuada para un amplio rango de protocolos basados en la especificación ASN.1.

La versión de este paquete utilizada en el desarrollo de la aplicación es ***python-pyasn1-0.0.11a-1***

Python-mysql

Es un adaptador para bases de datos MySQL para el lenguaje de programación Python. Está diseñado para aplicaciones multihilos que crean y destruyen gran cantidad de cursores y que ejecutan un gran número de INSERTs y UPDATEs concurrentes.

La versión utilizada en el proyecto es ***python-mysql 1.1.10***

M2Crypto

Esta librería es un adaptador completo de OpenSSL para el lenguaje de programación Python, la cual cuenta con: Algoritmos de Encriptación RSA, DSA, DH, HMAV, generación de digests (SHA-1), encriptación simétrica (como AES), funcionalidades SSL para implementar clientes y servidores.

Esta librería adicionalmente cuenta con dos módulos de gran utilidad para el desarrollo de este proyecto: El módulo EVP, el cual provee una interfaz de alto nivel para usar las funciones de encriptación de OpenSSL, y el módulo X.509, el cual permite la gestión, comprobación, edición y validación y firma de certificados X.509.

La versión utilizada es la ***M2Crypto 0.20.2***.

BitString

Es un módulo implementado íntegramente en Python que ayuda en la creación, análisis y modificación de datos binarios de la manera más simple y natural posible.

La versión utilizada es la ***bitstring2.0.3***.

Tkinter

Este módulo es una interfaz para las herramientas Tk para manejar la GUI (Interfaz Gráfica de Usuario) con Python. Está librería se ha utilizado en el desarrollo de las interfaces gráficas de los clientes de test.

La versión utilizada es la ***Tkinter.2.5.4***

Módulos propios de Python:

Adicionalmente, Python pone a nuestra disposición un conjunto de librerías más genéricas para el desarrollo de la aplicación, de las cuales las más empleadas en la aplicación son:

- Math: funciones matemáticas.
- Time: sincronización y timestamps.
- Random: números aleatorios.
- Threading: manejo de hilos
- Socket: implementación de sockets.
- Hashlib: funciones básicas de encriptación.
- Sys: datos de sistema.
- Os: datos del sistema operativo actual.
- Datetime: manejo de objetos de tipo fecha.

4.3 Arquitectura de la Aplicación

Durante el desarrollo de la aplicación, se han agrupado las clases desarrolladas en distintos paquetes de acuerdo a la funcionalidad que implementa cada una de ellas.

En la siguiente figura se puede observar la estructura de paquetes y directorios generados durante la implementación del Servidor de Revocaciones. A continuación detallaremos cada paquete generado y la funcionalidad que implementa cada uno de ellos:

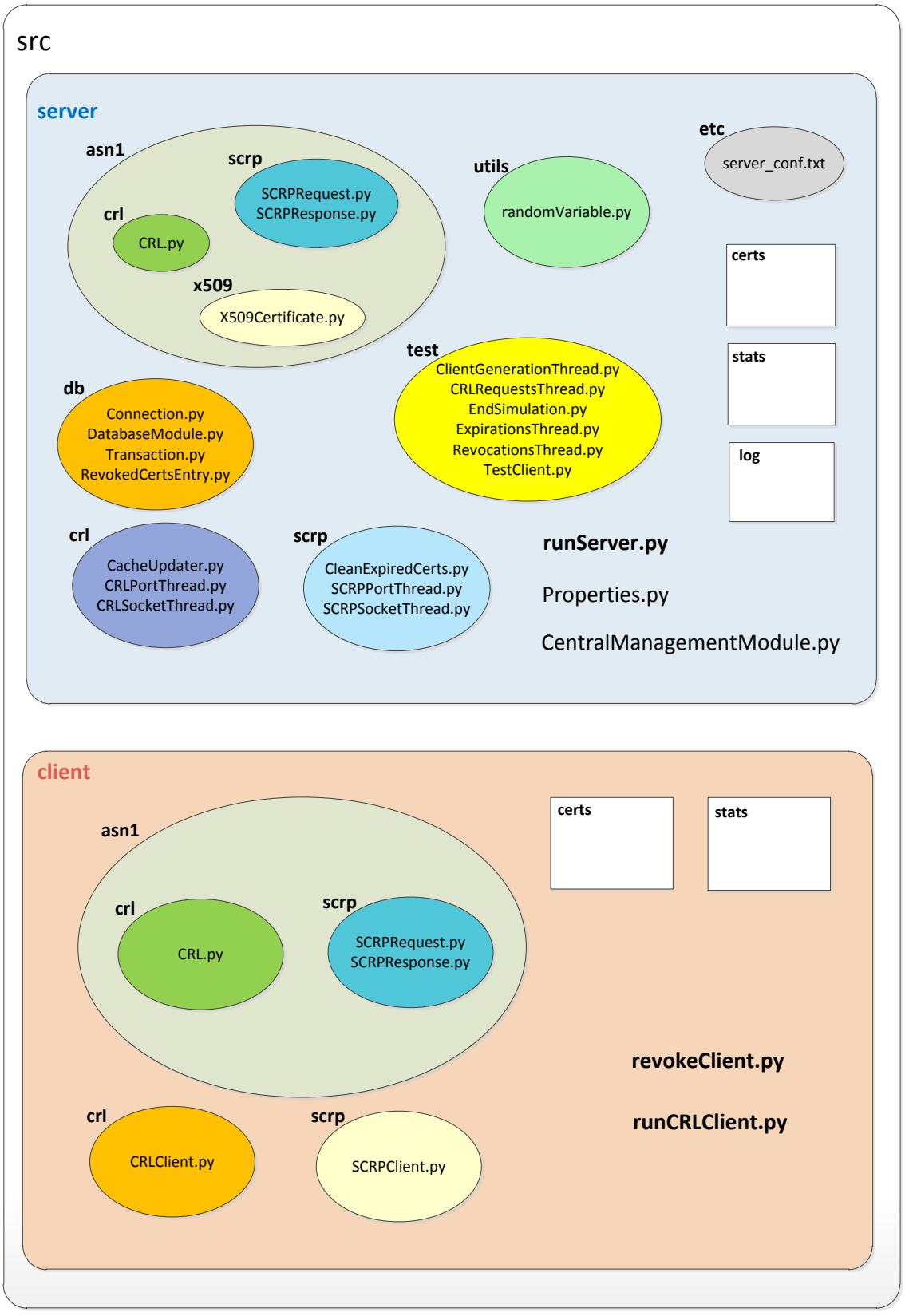


Figura 4.1. Arquitectura de la Aplicación.

SERVER

Paquete *asn1*

El paquete asn1 contiene las clases implementadas para servir de soporte a todos los paquetes de datos que intercambian tanto los clientes como el servidor en notación ASN.1.

La librería de ASN.1 para Python utilizada en este proyecto, no permite compilar las estructuras ASN.1, por lo que cada uno de los elementos ha tenido que ser parseado directamente de acuerdo a su definición ASN.1 indicadas en los RFCs correspondientes.

Dichas definiciones están añadidas como Apéndices al final de este documento.

De este modo, este paquete queda dividido en tres sub-paquetes:

- *x509:* Contiene la clase que describe la notación ASN.1 de un Certificado X509.
- *crl:* Contiene la clase que describe la notación ASN.1 de una CRL X509.
- *scrp:* Contiene las clases que describen la notación ASN.1 de una petición y una respuesta del protocolo SCRP.

Paquete *crl*

Contiene las clases que implementan la funcionalidad en el servidor para generar una CRL, y poder gestionar las solicitudes de descarga de dicha CRL por parte de los clientes. Consta de tres clases:

- *CacheUpdater*: es un hilo que cada vez que es ejecutado genera una nueva CRL (a partir de los datos de la Base de Datos) y la almacena en la caché del servidor.
- *CRLPortThread*: Clase que implementa un servidor de sockets: consiste en un socket creado escuchando peticiones de CRLs a un puerto determinado, al recibir la conexión de un cliente, crea un nuevo socket (clase CRLSocketThread) que permite llevar a cabo las transacciones entre el servidor y dicho cliente.

- *CRLSocketThread*: Es una instancia que implementa un socket donde están conectados un puerto del servidor y un cliente determinado. Esta clase es la que permite el envío de la CRL del servidor al cliente.

Paquete scrp

Contienen las clases que permiten enviar solicitudes y respuestas de revocaciones de certificados digitales. Para ello se han implementado:

- *CleanExpiredCerts*: Hilo que recorre la base de datos y elimina los certificados revocados que ya han expirado.
- *SCRPPortThread*: Implementa un socket que escucha peticiones SCRP de clientes, y crea un socket cliente con una conexión entre dicho cliente y un puerto del servidor.
- *SCRPSocketThread*: Socket que permite intercambiar peticiones y respuestas SCRP entre un cliente y el servidor.

Paquete db

Contiene las clases necesarias para permitir la conexión a la Base de datos, así como las distintas transacciones de base de datos soportadas por la aplicación.

- *Connection*: Clase que implementa la conexión a la Base de datos con el driver de MySQL para Python.
- *Transaction*: Implementa los distintos querys que se pueden ejecutar sobre Base de datos.
- revokedCertsEntry: clase que modelo la tabla de la base de datos del mismo nombre.
- DatabaseModule: Manejador de conexión, que gestiona el control de las otras clases pertenecientes a este módulo.

Paquete test

Contiene las clases que permiten ejecutar el Servidor de Revocaciones en modo TEST. En este modo, tanto la Generación de nuevos certificados, como las Revocaciones y Expiraciones, son generadas aleatoriamente siguiendo la distribución de una variable aleatoria exponencial.

Paquete stats

Contiene la clase Stats, que permite tomar una serie de estadísticas útiles para evaluar el Sistema de revocación implementado.

Paquete utils

Utilidades. Por el momento contiene únicamente la clase randomVariable, que permite generar variables aleatorias de distintas distribuciones sin utilizar las funciones propias provistas por Python.

CLIENT

El módulo de clientes contiene un paquete ASN.1 similar al del servidor, y dos paquetes adicionales: **crl** y **scrp.** Estos paquetes contienen las clases que implementan el cliente de test pata peticiones SCRP y solicitudes de descarga de CRL.

Adicionalmente existen dos scripts para la ejecución de la interfaz gráfica de cliente:

- revokeClient: permite ejecutar la interfaz gráfica de usuario para solicitudes de revocación de Certificados Digitales.
- runCRLClient: permite ejecutar la interfaz gráfica para simular un cliente que solicita la descarga de la CRL y para verificar si un certificado se encuentra revocado o no.

Tanto el código de la plataforma[5] como las instrucciones de uso[6] son especificados en los apéndices de este proyecto.

[5] **El código del proyecto se encuentra adjunto en el Apéndice C.**
[6] **El Manual de Uso de la Plataforma desarrollada también está adjunta en el Apéndice B.**

4.4 El Servidor

La plataforma desarrollada en este proyecto está dividida en dos partes: Servidor y Cliente.

La parte del Servidor se encarga de la gestión de las revocaciones, así como de la generación de la CRL y su distribución posterior a los usuarios que la soliciten. Desde el punto de vista lógico, la aplicación comprende cuatro módulos definidos: El módulo de Gestión de Revocaciones, el módulo de Base de datos, el módulo de Verificación de Estados (CRL) y el módulo de Administración Central.

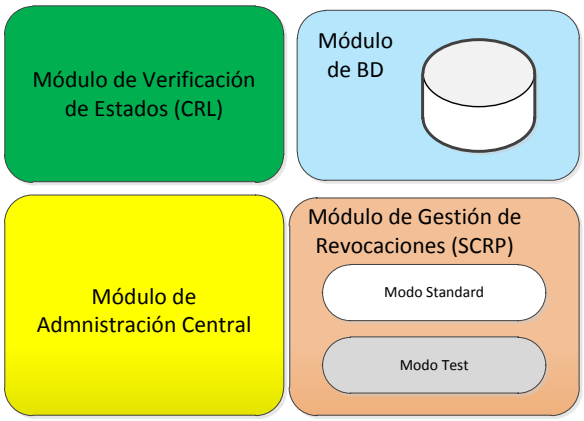


Figura 4.2. Modelo lógico de la aplicación.

Adicionalmente, para configurar los parámetros dentro de la simulación, existe un fichero editable: ***server_conf.txt*** en la carpeta ***etc*** del servidor.

Dicho fichero de configuración define los parámetros principales que determinan el comportamiento de la aplicación en la simulación a ejecutar. En la siguiente figura se presenta el fichero de configuración y sus principales parámetros configurables.

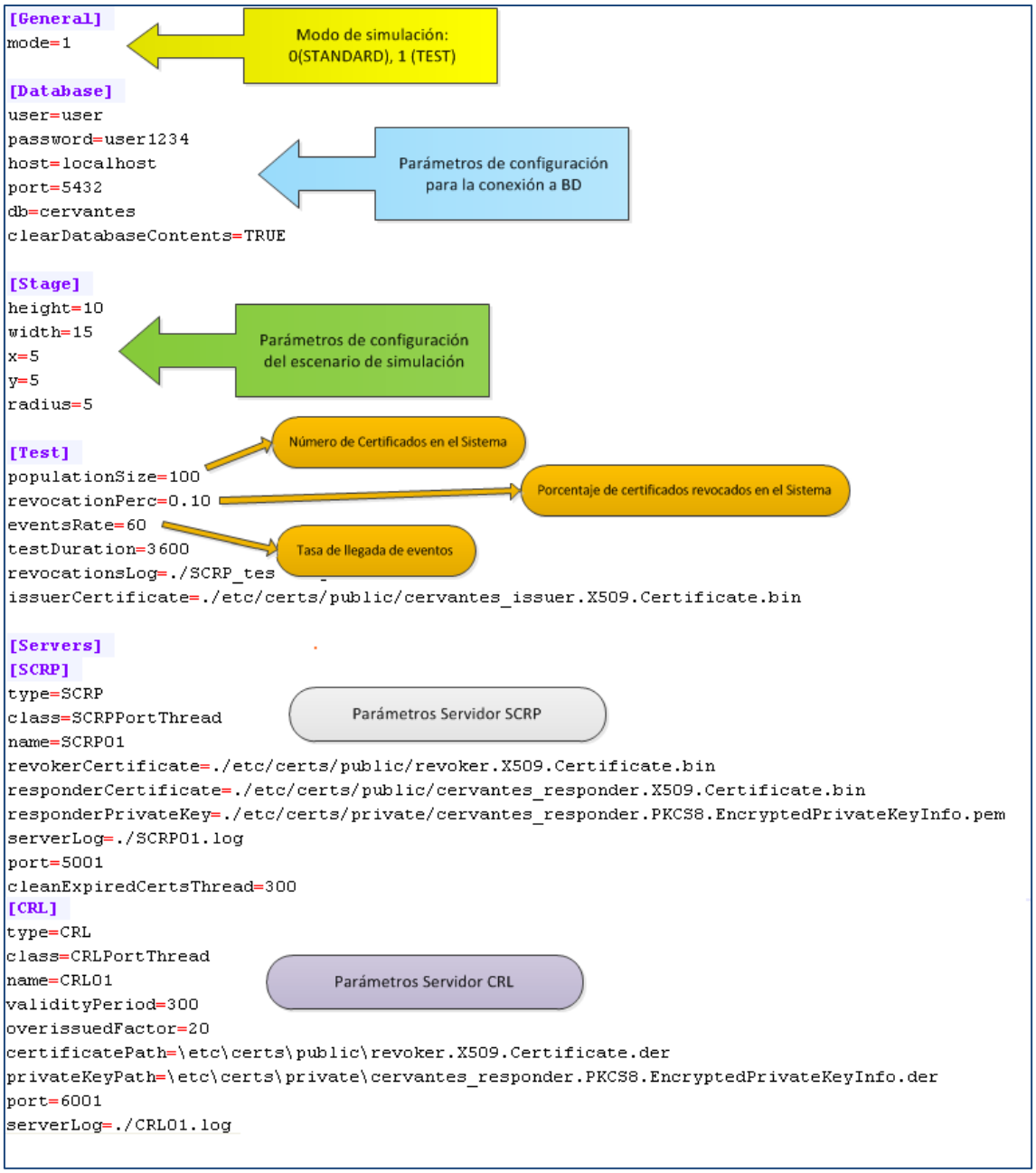


Figura 4.3. El archivo de configuración.

De estos parámetros cabe mencionar en esta sección el apartado **[Stage].** Para efectos de la simulación de conexiones y desconexiones de los usuarios se han aplicado dos soluciones:

- En la primera solución intervienen los algoritmos de movilidad de los clientes, y los parámetros definidos en *Stage.* Estos definen el escenario de la simulación, es decir, (x,y) son las coordenadas de ubicación del servidor, cuyo radio de cobertura es *radius*. Si un cliente en movimiento

se encuentra dentro de dicha zona de cobertura, estará conectado. Mientras se encuentra fuera de la zona de cobertura, se tendrá un estado de desconexión. En la siguiente figura podemos ver claramente este detalle.

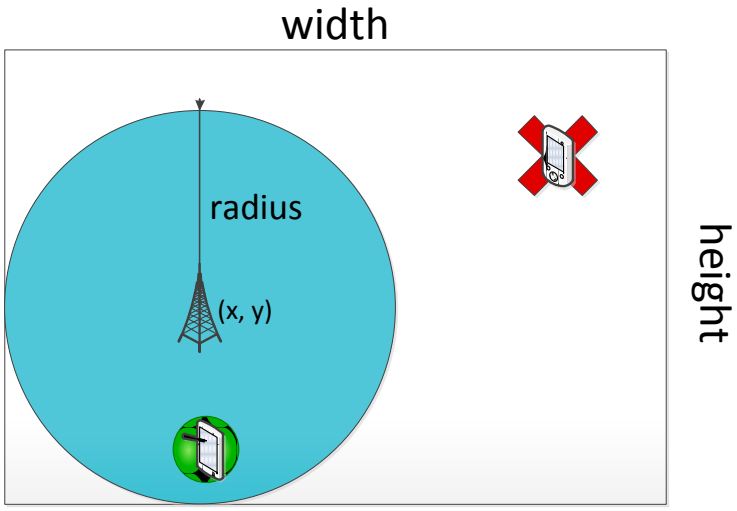


Figura 4.4. Definición del concepto Stage para simulación de conexiones/desconexiones.

- La segunda solución implica utilizar el algoritmo de Gilbert-Elliot para gestionar los estados de conexión/desconexión. Mediante este algoritmo, consideramos que los clientes cambian entre los estados de conexión y desconexión en intervalos largos de tiempo. Por tanto, cuando un cliente se desconecte, estará en este estado durante un intervalo de tiempo grande, por lo que es efectivo para obtener resultados en la evaluación del mecanismo del riesgo. Esta segunda solución es configurada enteramente en el lado del cliente, por lo que no es configurable desde el fichero de configuración del servidor.

4.4.1 Módulo de Base de Datos

Este módulo contiene las clases necesarias para permitir la interacción con la información contenida en la Base de datos. Esta base de datos es externa a la

aplicación, y contiene principalmente la información relacionada a los datos de estado de los certificados revocados en el sistema.

El modelo de datos es bastante simple y está compuesto por tres tablas:

- Issuers: Contiene la información acerca de los emisores de certificados.
- Revoked_certs: contiene la información del estado de los certificados que han sido revocados.
- Revocation_reasons: información acerca de los posibles códigos de revocación.

El paquete de base de datos implementa la clase DatabaseModule que permite efectuar operaciones sobre la base de datos, entre las cuales tenemos:

- Devolver todos los registros de la base de datos.
- Obtener el número de registros en la base de datos.
- Saber si un certificado está revocado o no.
- Eliminar todos los registros de la base de datos.
- Insertar un registro en la base de datos.
- Eliminar un registro de la base de datos.
- Listar todos los certificados expirados de la base de datos.
- Obtener un registro en particular.

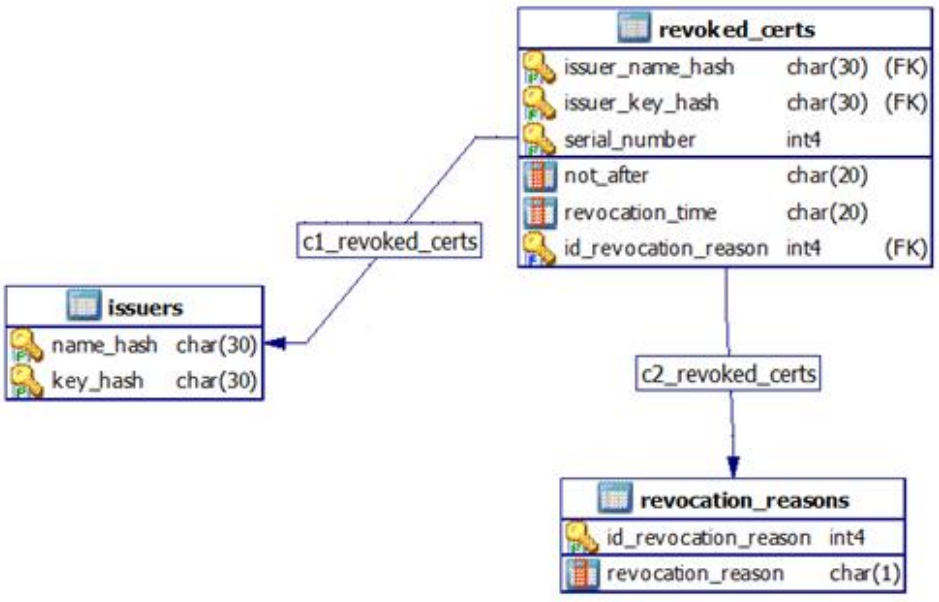


Figura 4.5. Modelo de datos.

4.4.2 Modulo de Gestión de Revocaciones

Este módulo se encarga de procesar las peticiones de revocación de certificados digitales. Este módulo se compone de un Servidor SCRP, el cual recibe las peticiones para la revocación de un certificado. Este se encarga de validar la petición y posteriormente de revocar el certificado, añadiendo un registro en la Base de datos con los datos de estado de dicho certificado.

Este módulo, encargado del proceso de revocación, puede funcionar en dos modos distintos: El modo estándar, el cual recibe peticiones SCRP, las procesa, y devuelve respuestas SCRP al cliente, y el modo test, donde los procesos de revocación, generación de clientes, y expiración son todos aleatorios y generados por el propio servidor.

El modo estándar

En el modo estándar se realizan todas las acciones que tiene lugar en un proceso real de Revocación de Certificados. La única diferencia es que en vez de usar el protocolo CMP, se utiliza el protocolo SCRP. Este protocolo es una versión reducida del protocolo CMP, que extrae de él las funcionalidades que son relevantes únicamente en el proceso de revocación. La descripción ASN.1 de este protocolo ***está añadida en los Apéndices de este proyecto.***

El protocolo SCRP funciona de la siguiente manera: El cliente SCRP envía una petición al servidor. Esta petición debe incluir el identificador único (serial_number) del certificado a ser revocado. Esta petición debe estar firmada por el cliente que la solicita.

Luego, el servidor SCRP verifica si el cliente está autorizado para llevar a cabo la revocación y envía el resultado de la revocación de vuelta al cliente.

La lógica con la que se ejecuta este modo, esto es, las tareas que tienen lugar en este módulo, se explican de manera simple en la siguiente figura:

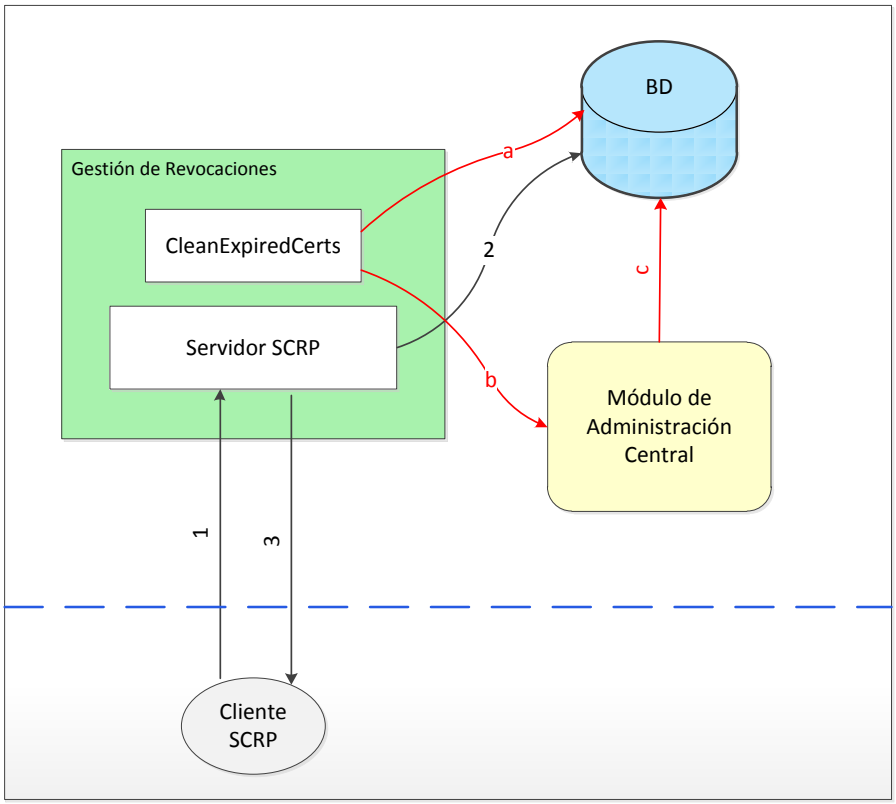


Figura 4.6. Lógica de implementación del Módulo de Gestión de Revocaciones.

1. Un cliente envía una solicitud SCRP de revocación de un certificado. En esta solicitud le debe indicar el certificado que se quiere revocar, y la razón de la revocación, así como el emisor de dicho certificado.

2. Una vez llega la petición al servidor, este verifica que el certificado no se encuentra ya almacenado en Base de datos (es decir, que no haya sido revocado previamente). Además verifica que el cliente esté autorizado para solicitar la revocación. Si todo esto es correcto, se procede a revocar el certificado añadiendo un registro en la Base de datos con los datos de estado del certificado. En caso que no se pueda realizar la revocación, o que el certificado ya haya sido previamente revocado, no se realiza ninguna operación y el servidor prepara una respuesta indicando el error ocurrido.

3. El servidor envía la respuesta con el código del resultado de la revocación al cliente SCRP, de manera que este pueda saber si se revocó correctamente el certificado o si ocurrió algún error.

Otras tareas que lleva a cabo este módulo son:

a. Existe un hilo creado, llamado ***CleanExpiredCerts*** que recorre la base de datos periódicamente, buscando registros de certificados ya expirados.

b. Una vez obtenidos los identificadores de los certificados expirados, se los envía al módulo de Administración Central.

c. El módulo de Administración elimina dichos registros de la Base de datos.

El modo test

El modo test trabaja de manera similar al estándar, con excepción de que las revocaciones y expiraciones de certificados son generadas aleatoriamente.

La generación aleatoria puede ser configurada por el usuario editando los parámetros del fichero de configuración *populationSize* (número de Certificados), *revocationPercentage* (porcentaje del total de certificados que estarán revocados) y *eventsRate* (tasa media de llegada de eventos).

La ejecución del modo test consiste en la ejecución de dos hilos: el generador de revocaciones y el generador de expiraciones.

Estos eventos aleatorios se dividen en dos fases:

Fase estática: en esta fase el generador de revocaciones trabaja a una tasa muy alta, de manera que se puedan alcanzar las condiciones indicadas en el fichero de configuración. En esta etapa el generador de expiraciones aún no está activo. El objetivo de esta fase es llenar la base de datos con registros de certificados revocados, y conseguir que el número de certificados revocados corresponda con el porcentaje indicado en el fichero de configuración. Los

resultados que se puedan observar para esta fase son irrelevantes, ya que constituyen el transitorio de la simulación.

***Fase dinámica**:* una vez alcanzadas las condiciones iniciales del sistema (configuradas en el fichero server_conf.txt), se inician los hilos de revocaciones y expiraciones con una tasa apropiada para mantener el sistema estable. Estos eventos se producen de acuerdo a los parámetros *eventsRate* (λ_{events}) y *revocationPercentage (r)* configurados.

Las tasas de revocaciones y expiraciones se definen como sigue:

$$\lambda_{\text{revocations}} = \frac{\lambda_{\text{events}}}{(1 - \text{r})}$$

$$\lambda_{\text{expirations}} = \frac{\lambda_{\text{events}}}{\text{r}}$$

En la aplicación desarrollada, se calcula el tiempo de espera para que ocurra el siguiente evento, ya sea de revocación o expiración, para poder dormir el hilo de ejecución durante ese tiempo. Al ser todos estos procesos variables aleatorias exponenciales, bastará con calcular el tiempo entre llegadas en un proceso aleatorio de distribución exponencial con media λ, de la siguiente manera:

$$t = -\frac{\lambda}{\ln(1 - \text{x})}, \text{x es una variable aleatoria uniforme entre } [0, 1]$$

En el modo test, el número de serie es generado eligiendo un certificado de manera aleatoria de entre toda la población de certificados.

4.4.3 Módulo de Verificación de Estados

Este módulo se encarga principalmente de:

- Generar periódicamente una lista que contenga todos los certificados que se encuentran revocados en el sistema (CRL).

- Mantener actualizada esta CRL de acuerdo a los cambios que puedan ocurrir (revocación de un nuevo certificado, expiración de un certificado previamente revocado).
- Facilitar la distribución de esta CRL, en el formato apropiado, a los clientes que soliciten descargarla.

En la siguiente figura se detalla la lógica del módulo de Verificación de Estados:

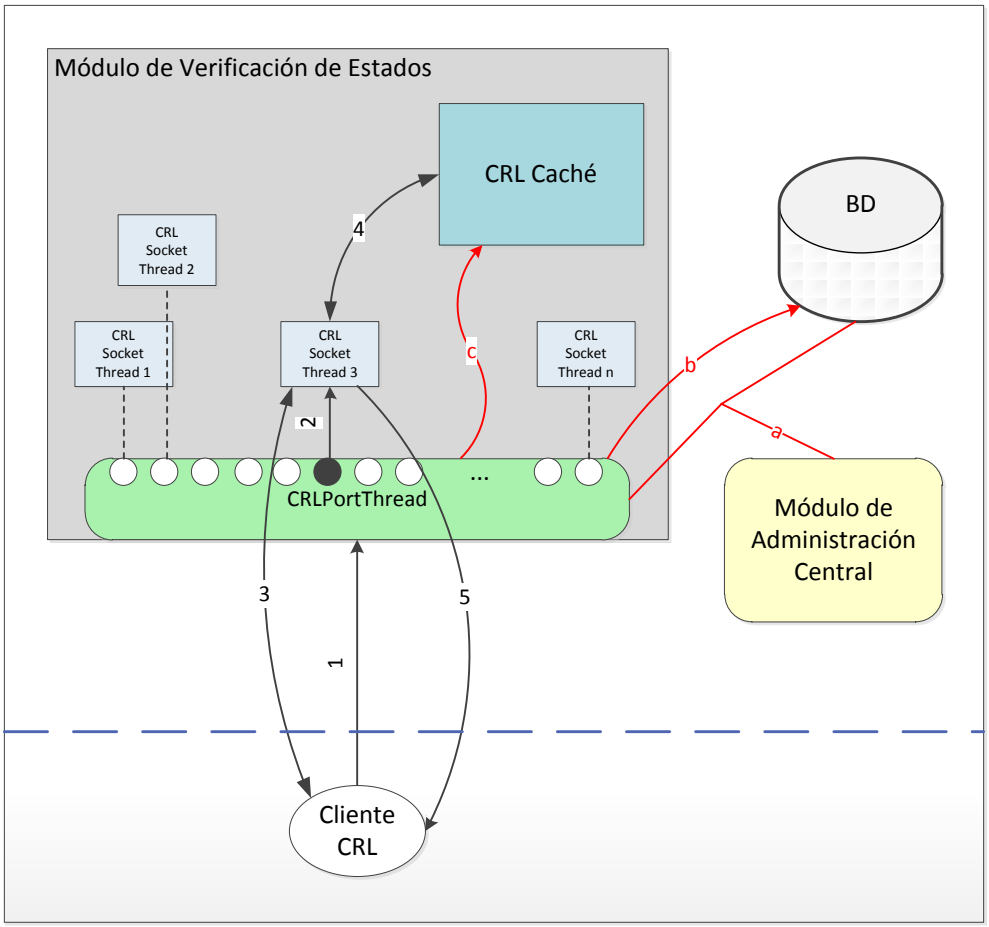


Figura 4.7. Lógica de implementación del módulo verificación de estados.

1. El cliente le solicita al servidor la descarga de la CRL, para conocer los certificados que se encuentran revocados.

2. Una vez recibida una solicitud, la clase CRLPortThread crea un socket entre el cliente que solicita la CRL y un puerto del servidor (este socket está definido en la clase CRLSocketThread).

3. De esta manera, el cliente queda conectado directamente con la instancia CRLSocketThread del servidor, para iniciar la descarga de la CRL.

4. Después de verificar que el cliente está autorizado para solicitar la CRL, el elemento CRLSocketThread pone a disposición del cliente la descarga de la CRL que tiene almacenada en caché.

5. El cliente procede a la descarga de la CRL, con lo que al recibirla tendrá la lista actualizada de los certificados revocados en el sistema.

Además, se llevan a cabo las tareas de actualización y mantenimiento de la CRL:

a. El módulo de administración central, a través de un hilo denominado CacheUpdater, verifica en base de datos si existen nuevas entradas de certificados revocados. En caso positivo, le notifica esto al controlador CRLPortThread de este módulo. Este hilo, *CacheUpdater,* no se vuelve a ejecutar hasta que termine el tiempo de validez de la CRL.

b. El controlador *CRLPortThread* crea una nueva CRL a partir de los registros existentes en la Base de datos.

c. Se guarda la nueva CRL generada en la caché del servidor, para que los clientes pueden descargarla desde allí.

Mediante este módulo, todos los clientes activos del sistema pueden saber, de la manera más precisa posible, qué certificados se encuentran revocados en un momento determinado. Sin embargo, cuando un cliente entra en estado de desconexión, no podrá descargar la CRL del servidor, por lo que la información que tenga en la última CRL que haya descargado puede no ser completamente fiable.

En este sentido, cabe mencionar que el mecanismo del riesgo es una solución que se propone para este tipo de problemas: podemos calcular, valiéndonos de métodos estadísticos, cuál sería el riesgo de operar con un certificado cuando

no estamos seguros de su validez, debido a que no disponemos de información actualizada de los certificados revocados en el sistema. De acuerdo al valor del riesgo obtenido, y a la elección de un umbral de riesgo, el cliente podrá decidir si dar por válido o no el certificado, asumiendo una cierta probabilidad de error.

Este proyecto no se enfoca en implementar el Mecanismo de Emisión con Riesgo Contante (CRI), más bien, lo que se hace es evaluar y calcular el valor real del riesgo. El cliente calcula el riesgo real de operar con un certificado cuando se encuentra en estado de desconexión. Todos los datos que se manejan son reales, no son calculados mediante métodos estadísticos.

Para esto, el cliente debe calcular los conjuntos de certificados necesarios para obtener el valor del riesgo: Número de Certificados Revocados Desconocidos (U) y Número de Certificados Operativos (O). Sin embargo, el cliente no dispone de la información necesaria para calcular estos conjuntos. Dicha información solo es conocida por el Servidor en todo momento.

Para que los clientes puedan disponer de todos los datos reales en cualquier momento, el Servidor tiene implementada una Api[7] que le permite al cliente consultar el valor de los datos que le hagan falta directamente al servidor. Los datos requeridos para la toma de estadísticas y evaluación del riesgo se detallan en el siguiente capítulo del proyecto.

4.4.4 Módulo de Administración Central

Este módulo se encarga de inicializar el servidor, y de gestionar el funcionamiento de los otros módulos de manera ordenada y correcta.

Las tareas que lleva a cabo durante el proceso de inicialización del servidor son las siguientes:

1. Inicializar módulo de estadísticas: sondeo del número de certificados en el sistema y del número de certificados revocados en el servidor.

[7] **Esta Api no forma parte de un Servidor de Revocaciones Real. Únicamente ha sido desarrollada en este proyecto a fines de ser una herramienta para la evaluación del algoritmo del riesgo.**

2. Leer las propiedades definidas en el fichero de configuración del servidor.
3. Determinar el modo de ejecución de la aplicación: STANDARD o TEST.
4. Inicializar la Base de Datos del Sistema. Si en el fichero de configuración se indica que se debe limpiar la Base de Datos (*clearDatabaseContents=TRUE*), se eliminan todos los registros previamente existente en la Base de datos.
5. Configurar los puertos del servidor que se encargarán de escuchar las peticiones de los clientes: Un puerto para solicitudes de revocación SCRP, y otro para solicitudes de descarga de CRL.
6. Si el modo de ejecución es TEST, preparar la ejecución de los hilos de revocaciones, expiraciones y generaciones aleatorias.
7. Inicializar el API de estadísticas del servidor (para que los clientes puedan consultar datos del servidor).
8. Inicializar el Módulo de Revocaciones: En el modo STANDARD el servidor se pondrá en modo de escucha esperando las peticiones de revocación de certificados. En el modo TEST, inicia la ejecución de los hilos de generación aleatoria de revocaciones, expiraciones y creación de nuevos clientes.
9. Inicializar el Módulo de Verificación de Estado: Cada cierto tiempo (*validityPeriod* en el fichero de configuración del servidor), se generará una CRL nueva a partir de la información contenida en la Base de Datos de Revocaciones de la Plataforma, para que los clientes puedan descargarla.
10. El servidor comenzará a escuchar, en los puertos especificados, las peticiones de los clientes.

Durante la ejecución de la aplicación, se encarga de gestionar:

- Los procesos de revocaciones y expiraciones.
- Consultar, insertar y eliminar registros de la Base de datos.
- Controlar la ejecución de toma de estadísticas.
- Finalizar todos los hilos de ejecución, cuando la simulación haya terminado.

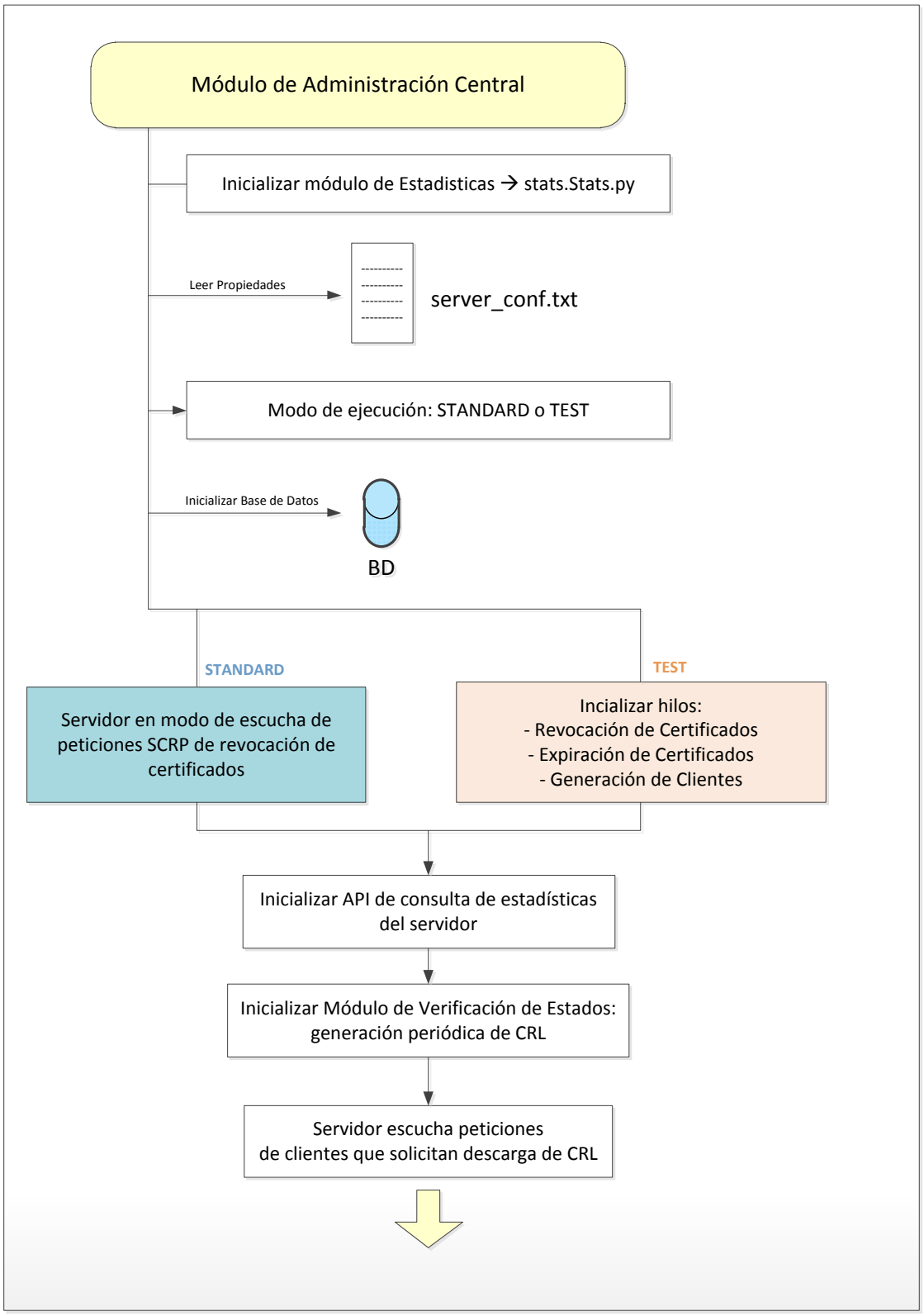


Figura 4.8. El Módulo de Administración Central.

4.5 Los Clientes

La segunda parte de la plataforma corresponde al módulo de los clientes. En esta plataforma pueden existir únicamente dos tipos de clientes: Aquellos que solicitan la revocación de un certificado (Clientes SCRP), y aquellos que solicitan la descarga de una CRL (Clientes CRL).

Para ambos tipos de clientes, se ha desarrollado una sencilla interfaz gráfica que permite realizar las peticiones de acuerdo a parámetros configurables en la propia interfaz.[8]

Cliente SCRP

Los clientes SCRP permiten enviar una solicitud, *SCRP Request,* para pedir la revocación de un determinado certificado. La interfaz gráfica permite seleccionar distintos parámetros necesarios para realizar esta solicitud de revocación, los cuales son:

- *Responder URL*: Es la URL a la cual se hace la petición de revocación. Debe tener la siguiente estructura: *http://<ip_address>:<port>, donde ip_address* indicará la dirección IP donde se encuentra el servidor, y *port* será el puerto que tiene configurado el servidor para escuchar y procesar las peticiones de tipo SCRP.
- *Identity File:* Debe contener la ubicación del fichero donde se encuentra guardada la clave privada con la cual será firmada la solicitud SCRP.
- *Responder Certificate:* Debe contener la ubicación del fichero donde se encuentra el certificado digital de la entidad encargada de responder a la solicitud de revocación. En otras palabras, si la respuesta es enviada por el mismo servidor, debe contener la ubicación del certificado digital del servidor.

[8] **El Manual de Uso de las interfaces GUI para los clientes está detallado en el Apéndice B de este proyecto.**

- *Issuer of Certificate:* Ubicación del certificado digital del emisor del certificado que se quiere revocar.
- *Certificate to Revoke:* Ubicación del certificado digital del cliente que se quiere revocar.

Adicionalmente, esta interfaz permite generar una revocación de un certificado el cual podemos definir en el momento, es decir, se toma como base un certificado por defecto (*default.pem*) y se requerirá tres parámetros para definir sobre este certificado: El número de serie, la fecha de expiración y la razón de revocación.

Un cliente SCRP funciona de la siguiente manera:

- El cliente se conecta al servidor, enviando al puerto habilitado para escuchar las peticiones de revocación (definido en los parámetros de configuración del sistema) una petición usando el protocolo SCRP (*SCRP Request*).
- El servidor recibe la petición, la valida (comprueba que el cliente esté autorizado a solicitar la revocación, y que el certificado a revocar no tenga ya un registro en la Base de datos), revoca el certificado y devuelve al cliente el resultado de la revocación (*SCRP Response*).
- El cliente recibe la respuesta SCRP y podrá observar el resultado de la revocación solicitada.

Cliente CRL

La interfaz gráfica del cliente CRL permite descargar una CRL, visualizar la CRL descargada, verificar si un certificado en particular se encuentra revocado o no, y simular la ejecución de un hilo de un cliente.

Sin embargo, también existe un cliente CRL el cual puede ser ejecutado por línea de comandos, que permite iniciar la ejecución de un hilo que controla a un cliente CRL. Esto es, descarga la CRL y cuando su CRL haya expirado, intentará conectarse al servidor para descargar la nueva actualización. Sin embargo, aquí también debemos tener en cuenta que estos clientes pueden caer en periodos de conexión y desconexión de duración variable. Si el cliente se encuentra en estado de desconexión, no podrá enviar peticiones de CRL al

servidor hasta que recupere la conexión con el servidor. El funcionamiento de un cliente CRL es:

- El cliente envía una solicitud de descarga de CRL, enviando, entre los datos, su identificador único para determinar si no se encuentra revocado (en caso de estar revocado, no podrá solicitar la CRL). Esta petición se envía a un puerto específico del servidor, donde se escuchan exclusivamente solicitudes de descarga de CRLs.
- El servidor recibe la petición, valida al cliente y crea un nuevo socket entre el cliente y el servidor, para poder realizar la transferencia de la CRL.
- El servidor envía la CRL que tiene almacenada en caché al cliente.
- El cliente recibe la CRL y la almacena en su caché local.

Sin embargo, aparte de estas tareas, los clientes CRL realizan dos acciones sumamente importantes:

1. Gestión de conexiones/desconexiones

   Como estamos simulando un entorno con presencia de interrupciones (entorno móvil, VANET, DTN), debemos definir una manera para simular las conexiones intermitentes en los clientes. Para ello se han desarrollado dos mecanismos:

   - El primero consiste en añadir el concepto de movilidad en los usuarios, esto es, el usuario tendrá un hilo que se ejecutará periódicamente actualizando la posición del cliente en base a una velocidad y una posición inicial definida durante la creación del usuario. Para este mecanismo se utilizan dos métodos, un método de caminos aleatorios (Random Waypoint Model) y otro en base a un modelo estadístico (Modelo Gauss-Markov). Aquí se introduce la idea del Escenario (Stage) y Zona de Cobertura, explicados en el apartado 4.1 de este capítulo, cuyos parámetros son definidos, como se ha visto, en el archivo de configuración. La lógica es sencilla: Mientras el cliente esté dentro de la Zona de Cobertura, tendrá conectividad con el servidor, y mientras esté fuera de ella, se tendrá un estado de desconexión.

- El segundo mecanismo consiste en el uso del modelo Gilbert-Elliot con dos estados: conexión y desconexión. Este modelo está caracterizado por una cadena de Markov de 2 estados ($\{C_n\}_{n\geq 0}$, con N=2), donde C_n es una variable aleatoria asociada al estado del canal en el instante de tiempo n. La probabilidad de transición del estado "Conectado" (C) al estado "Desconectado" (D) es PCD. De la misma manera, PDC es la probabilidad de transición de D a C. Las probabilidades de estado estacionario de estar en el estado C o D son:

$$\pi_D = \frac{P_{CD}}{P_{CD} + P_{DC}}$$
$$\pi_C = \frac{P_{DC}}{P_{CD} + P_{DC}}$$

2. Toma de estadísticas

Cada cliente tiene implementado un hilo que se ejecuta periódicamente para imprimir en un fichero externo información acerca del estado de los certificados desde el punto de vista de cada uno de los clientes. Esto es, por cada cliente existente, se generará un fichero de salida con esta información. La información que contendrá corresponde a los distintos conjuntos de certificados desde el punto de vista del usuario, explicados en el capítulo anterior de este documento, para la evaluación del mecanismo del riesgo. Estos datos son:

- Número de Certificados No Expirados (C).
- Número de Certificados Operativos (O).
- Número de Certificados Revocados (V).
- Número de Certificados Revocados Conocidos (K).
- Número de Certificados Revocados Desconocidos (U).
- Número de Certificados Buenos (G).
- Cálculo del valor del Riesgo.

Estas estadísticas son de gran importancia, ya que son las que nos permitirán evaluar el valor del riesgo existente en la emisión periódica de CRLs en entornos de baja conectividad.

Capítulo 5

SIMULACIÓN Y EVALUACIÓN DE RESULTADOS

5.1 Introducción

Una vez que tenemos implementada la Plataforma de Revocación de Certificados Digitales, hace falta verificar el correcto funcionamiento de la misma.

Para ello, realizaremos una serie de simulaciones con la plataforma implementada que nos permitan verificar que su comportamiento básico es correcto, es decir, que gestione correctamente tanto las peticiones de revocación como las solicitudes de descarga de CRL. Además debemos de verificar la correcta toma de estadísticas que nos permitirán posteriormente calcular el riesgo.

En este capítulo nos remitiremos a indicar los parámetros de configuración de la simulación y a mostrar los resultados obtenidos gráficamente. Los pasos a seguir para ejecutar la simulación en la plataforma, así como las instrucciones para configurar e inicializar el servidor y los clientes, son detallados en el Apéndice B.

5.2 Casos de Simulación

El primer caso de simulación que ejecutaremos nos deberá permitir verificar el correcto funcionamiento del servidor. Para ello debemos hacer el seguimiento de la Evolución de los Certificados Existentes en el Sistema, así como la Evolución de los Certificados Revocados en el Sistema. Podemos graficar estos datos y observar como se comporta realmente el servidor durante el tiempo de ejecución.

Otro caso de simulación nos deberá permitir visualizar el Ancho de Banda de Utilización del Servidor, y en este sentido, verificar el uso de las CRLs sobre-emitidas. Variaremos el número de CRLs que se emitirán en cada periodo de validez, de manera que el Ancho de Banda de Utilización quede más distribuido y evitemos intervalos de saturación del servidor.

El paso siguiente será verificar que las estadísticas tomadas por cada usuario permitan calcular correctamente el valor del riesgo. Básicamente, las estadísticas que debemos tener en cuenta son:

- Número de Certificados Emitidos No Expirados (*C*).
- Número de Certificados Operativos (*O*).
- Número de Certificados Buenos (*G*).
- Número de Certificados Revocados (*V*).
- Número de Certificados Revocados Desconocidos (*U*).
- Cálculo del valor del riesgo real: $riesgo = \frac{U}{O}$

Para comprobar esto, requerimos de dos simulaciones que nos permitirán calcular y graficar la evolución de estos conjuntos, así como la evolución del riesgo: Evaluaremos la evolución del riesgo en la emisión periódica de Información de Estado de Certificados (CSI) en un entorno con muy pocas interrupciones (Probabiliidad de Conexión: P_C = 0.95), y en un entorno donde haya conectividad baja (P_C = 0.15).

De esta manera veremos la evolución del riesgo en estos dos escenarios, y nos concentraremos en buscar los valores de configuración necesarios para

obtener los mejores resultados que nos permitan apreciar el correcto funcionamiento del mecanismo de toma de estadísticas y cálculo del riesgo implementado por esta plataforma.

5.3 Evaluación de Ancho de Banda de Utilización y la Evolución de los Certificados en el Sistema

En esta primera simulación, verificaremos simplemente que el el Servidor de Revocaciones funcione correctamente. Para esto, generaremos las revocaciones de certificados, las expiraciones y la creación de nuevos clientes de manera aleatoria. Es decir, utilizaremos la plataforma en modo TEST.

Graficaremos el Número de Certificados Emitidos No Expirados en el Sistema (*C*), y el Número de Certificados Revocados No Expirados en el Sistema (*V*).

Debemos verificar en dichas figuras que los conjuntos evaluados si bien varían ligeramente a lo largo de la simulación, se mantienen estables fluctuando alrededor de su valor definido en los parámetros de configuración.

Los parámetros más relevantes en esta experiencia son:

Existirán mil certificados en el sistema (*numberOfCertificates = 10000*), el tiempo de espera entre llegadas de eventos será de 10 segundos (*eventsRate = 10 min,* $\lambda_{eventos}$ *= 6 eventos/hora*) y el porcentaje de revocación será del 10% (*revocationPerc = 0.1*). Además el periodo de validez de una CRL será de 4 horas (*validityPeriod = 240 min*) y el número de CRLs emitidas por cada periodo de validez será 1 (*overissuedFactor = 1*). La simulación se ejecutará durante 18000 minutos (12.5 días).

De estos datos tenemos que la tasa de llegada de eventos de revocación y de eventos de expiración será:

$$\lambda_{revocaciones} = \frac{10}{(1-0.1)} = 11.11\ min \quad \rightarrow 5.4 \text{ revocaciones/hora}$$

$$\lambda_{expiraciones} = \frac{10}{0.1} = 100\ min \quad \rightarrow 0.6 \text{ revocaciones/hora}$$

Además, deducimos que en media cada 11.11 minutos llegará una nueva solicitud de revocación, y cada 100 minutos expirará un certificado. No se usarán CRLs sobre-emitidas, ya que solo se emite una CRL durante cada periodo de validez.

Los resultados obtenidos son los siguientes:

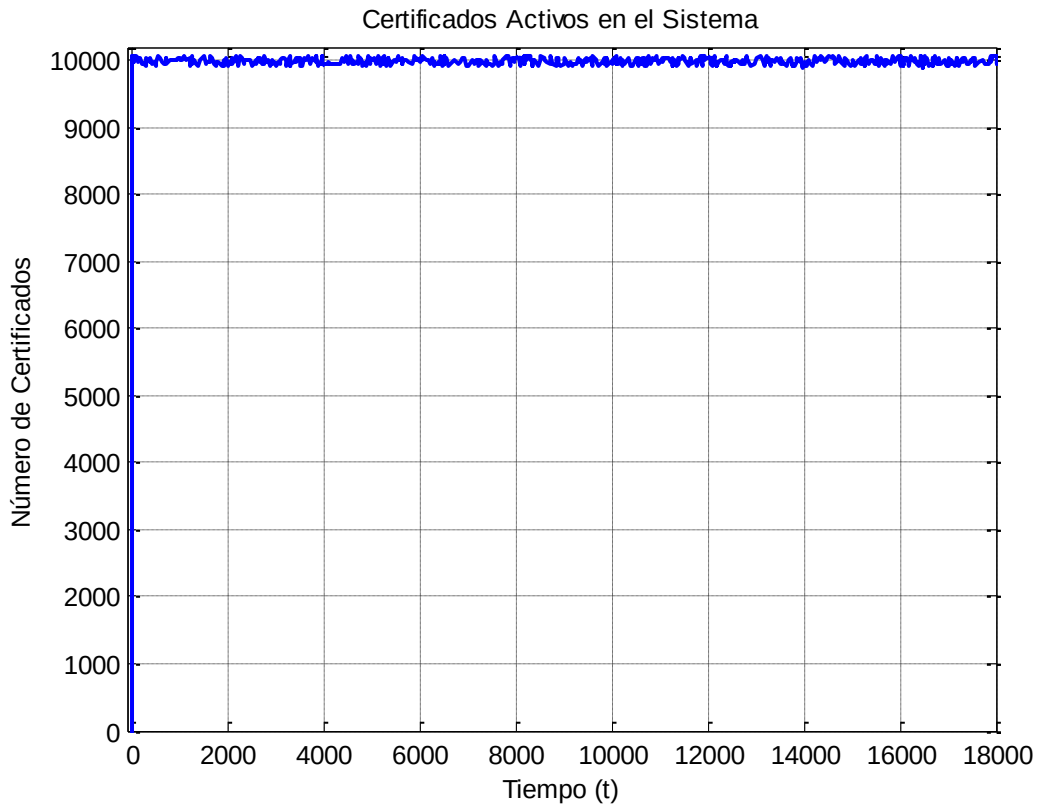


Figura 5.1. Evolución del Número de Certificados No Expirados en el Sistema.

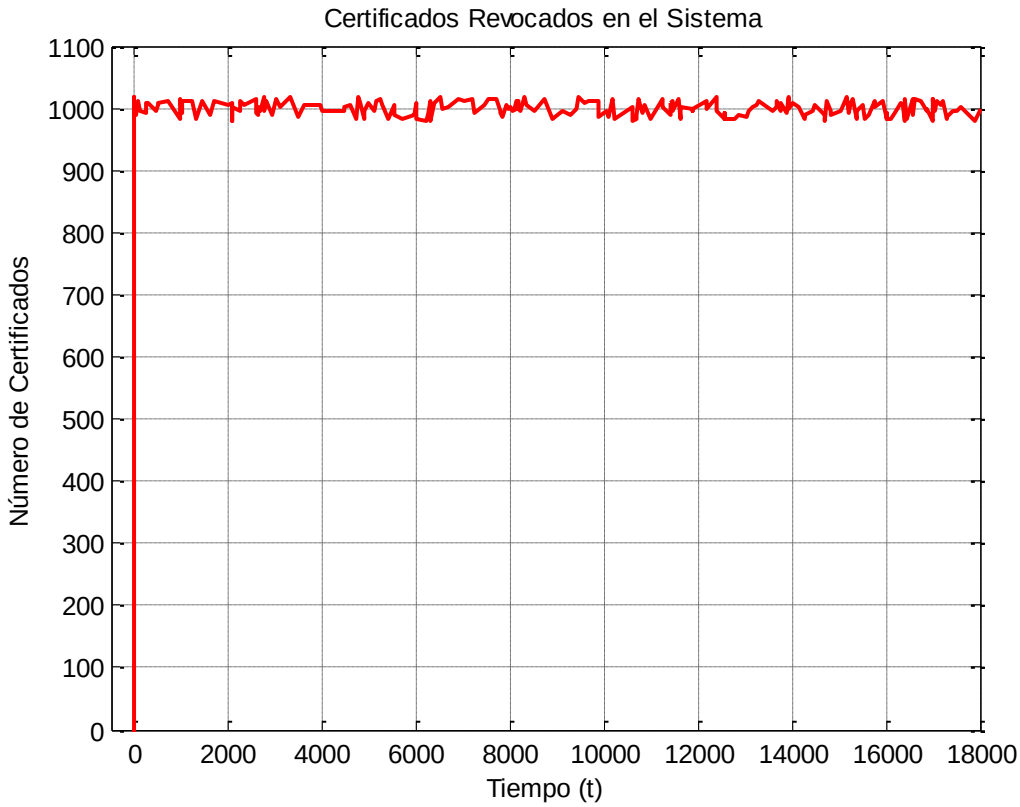


Figura 5.2. Evolución del Número de Certificados Revocados en el Sistema.

Como podemos apreciar en la Figura 5.1, los Certificados Emitidos No Expirados en el Sistema varían ligeramente alrededor del valor de 10000 certificados. Como el procentaje de revocación es de 10%, en la Figura 5.3 podemos apreciar que el Número de Certificados Revocados varía alrededor del valor de los 1000 certificados.

Esto nos permite apreciar que la generación aleatoria de eventos por parte del servidor funciona correctamente y que su comportamiento se mantiene estable dentro de los parámetros configurados.

A continuación, evaluaremos el Ancho de Banda de Utilización del Servidor. Esto nos mostrará la cantidad de Bytes que se encuentra transmitiendo el Servidor a lo largo del tiempo. Con los parámetros configurados en la experiencia indicada arriba, podemos apreciar que la utilización del servidor, sin el uso de la CRL sobre-emitida, tiene la siguiente forma:

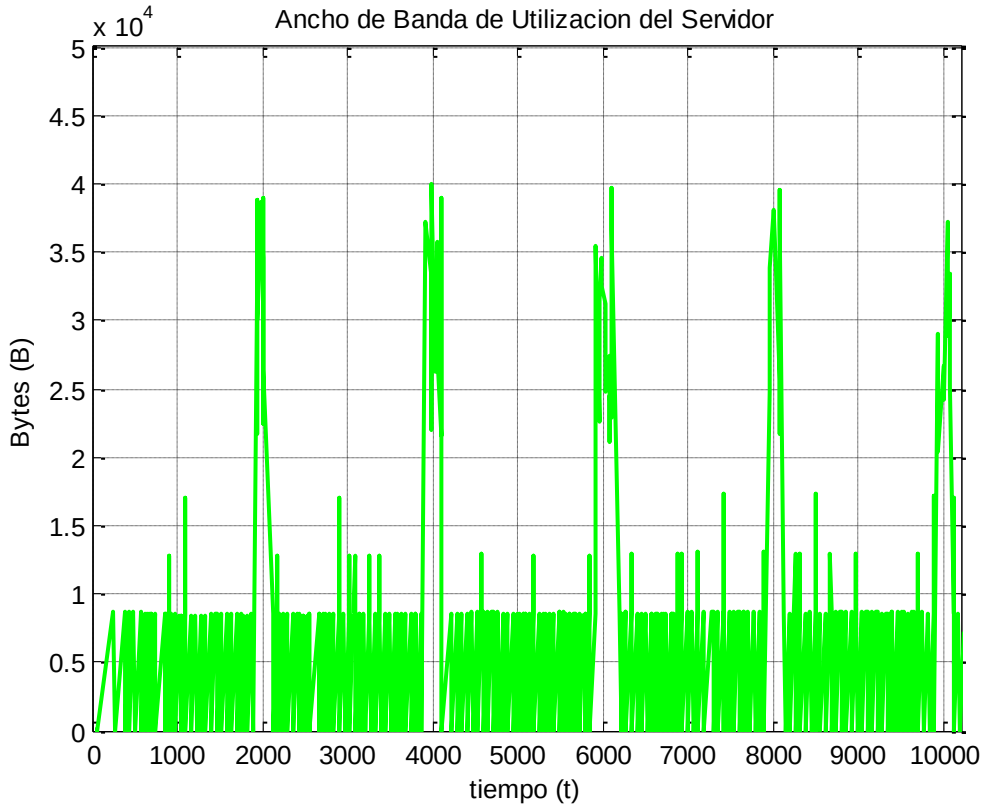


Figura 5.3. Ancho de Banda de utilización del Servidor usando CRL Tradicional (O = 1)

Se puede notar que aproximadamente cada 2000 minutos se genera un pico de solicitudes de descarga de CRL. Como todos los usuarios tendrán la misma CRL con un mismo valor de *nextUpdate,* todos intentarán descargar la nueva CRL disponible al mismo tiempo.Esto ocurre al terminar el periodo de validez de la CRL. Como solo se emite una CRL durante cada intervalo, todos los clientes enviarán su solicitud de descarga, por lo que obtenemos picos que muestran la descarga hecha por múltiples usuarios, puediendo causar la saturación del Servidor debido a la gran cantidad de descargas concurrentes.

Por último, ejecutaremos la simulación nuevamente utilizando los mismos parámetros, excepto que esta vez utilizaremos el mecanismo de CRL sobre-emtida. Durante cada intervalo de validez emitiremos 4 CRLs en tiempos distintos, todas ellas con una validez igual al periodo de validez general. Esto se consigue editando el fichero de configuración:

validityPeriod = 2000, overissuedFactor = 4

De este modo se emitirán 4 CRLs por cada intervalo de validez, de manera que los clientes que soliciten la descarga de la CRL podrán tendrán diferentes valores en el campo *nextUpdate* dependiendo de cual de las 4 CRLs haya descargado, por lo que en vez de que todas las peticiones queden concentradas en un solo momento, estarán repartidas en 4 bloques, aprovechando de una manera más eficiente el Ancho de Banda de Transmisión del Servidor y evitando periodos de saturación por la concentración de muchas descargas concurrentes.

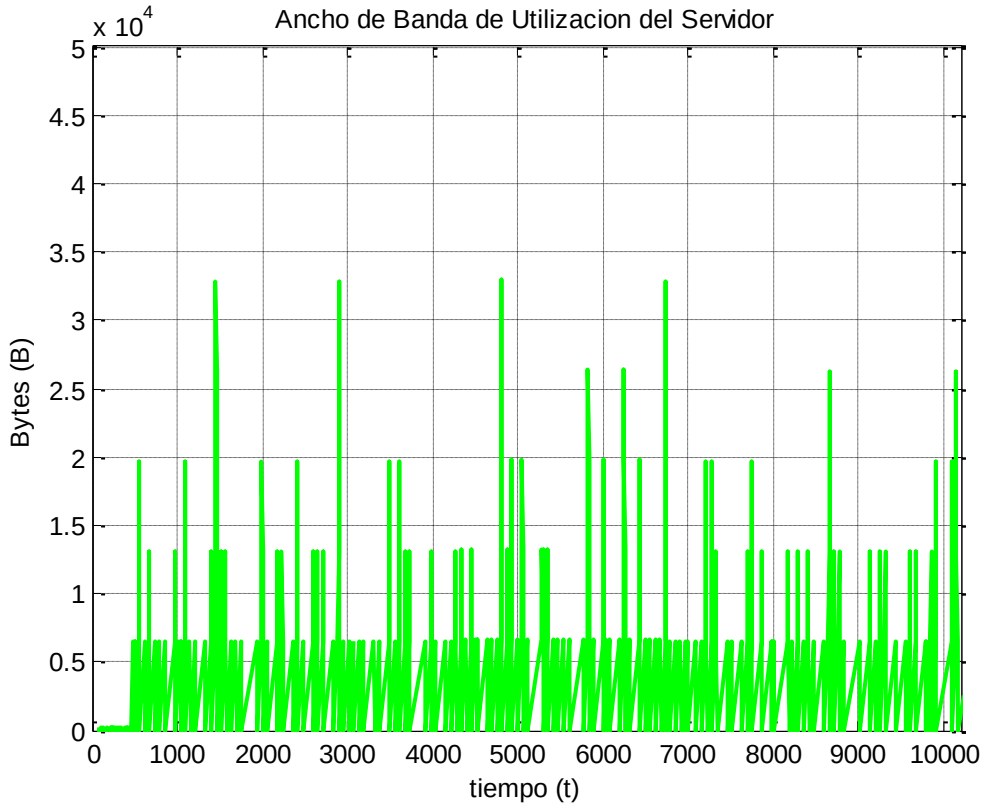


Figura 5.4. Ancho de Banda de Utilización del Servidor usando CRL sobre-emitida (O = 4)

Aquí podemos apreciar que cada 2000 minutos, se generan 4 picos de solicitudes de descarga de CRLs. De este modo, tenemos que el tráfico en el servidor queda dividido en 4 porciones, cuyos picos de utilización son más pequeños. Así aprovechamos mejor el servidor evitando que caiga en periodos de desuso, de baja actividad o de excesiva saturación.

5.4 Evaluación del Riesgo en la Emisión periódica de Información de Estado de Certificados en un entorno sin interrupciones

En este segundo caso, asumiremos que el entorno presenta un alto nivel de conectividad (Probabilidad de Conexión P_C = 0.95). Esto significa que cada cliente estará conectado el 95% del tiempo, de modo que las desconexiones son muy cortas (y en algunos casos inexistentes), por lo que se podrá descargar la CRL sin problemas tan pronto sea actualizada en el servidor.

Los demás parámetros los definimos como sigue:

- El Número de Certificados en el Sistema será 1000.
- El Porcentaje de Revocacion *p* será de 40%.
- Tiempo entre llegada de eventos: 10 minutos.

$$revocationsRate = \frac{10}{1 - 0.4} = 16.66\, min$$

$$expirationsRate = \frac{10}{0.4} = 25\, min$$

- Periodo de validez de CRL: 240 minutos.
- CRL Sobre-emitida: Sí (se emitirán 4 CRLs por periodo de validez).
- Tiempo de Simulación: 18000 minutos = 12.5 días
- Probabilidad de Conexión del Cliente: P_C = 0.95, P_D = 0.05

Las gráficas obtenidas a partir de los resultados son:

La evolución del los certificados en el sistema se mantiene estable alrededor del valor de 1000 certificados:

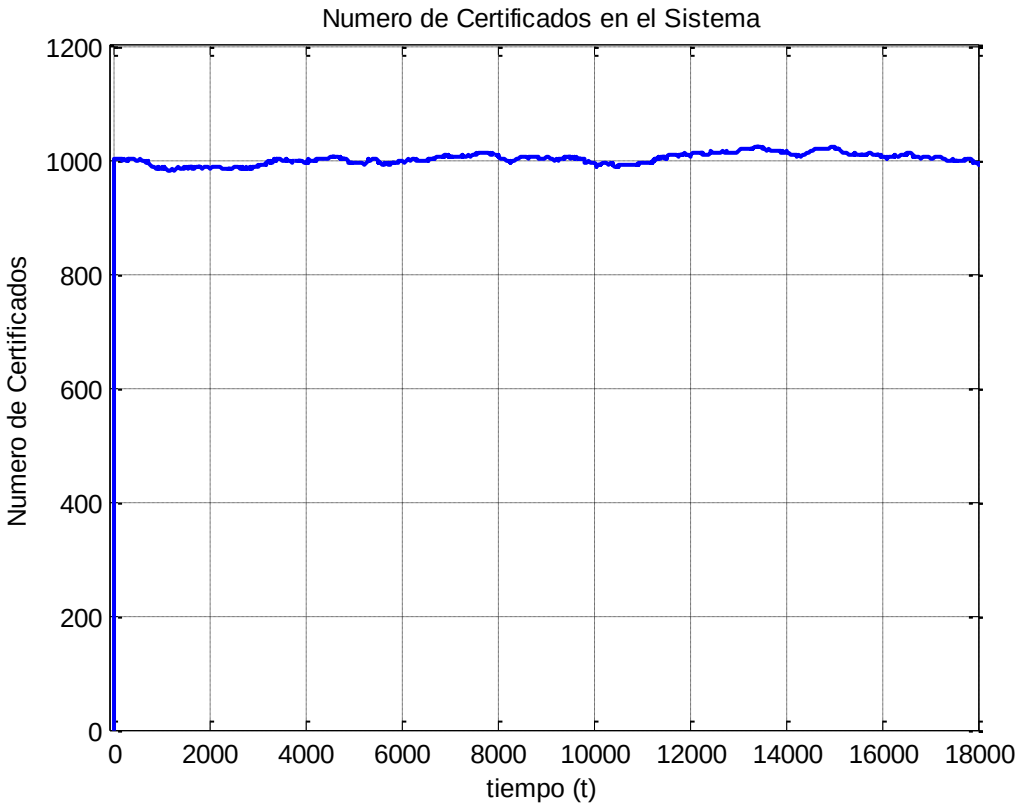


Figura 5.5. Evolución del Número de Certificados Emitidos No Expirados en el Sistema en un entorno sin interrupciones.

La evolución del los certificados revocados en el sistema se mantiene estable alrededor del valor de 400 certificados (p = 0.4):

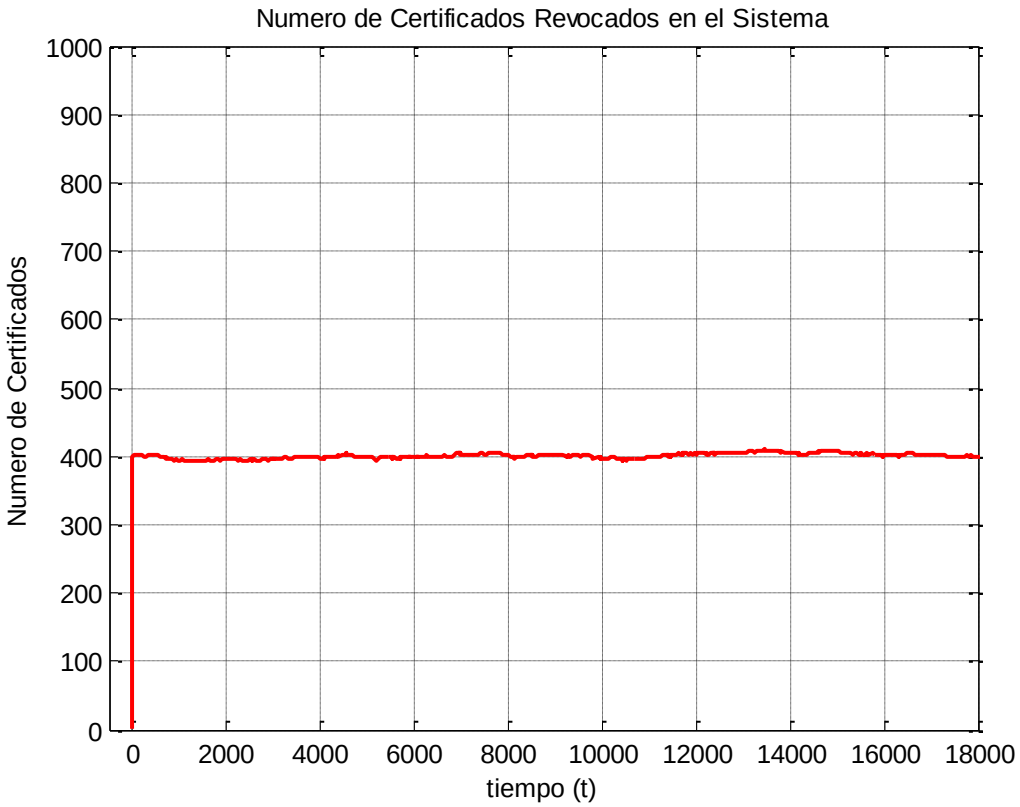


Figura 5.6. Evolución del Número de Certificados Revocados en el Sistema en un entorno sin interrupciones.

De este modo, podemos graficar la variación del Porcentaje de Certificados Revocados en el Sistema (*p*), y podemos apreciar, como es esperado, que se mantiene estable alrededor del valor de 0.4 (40% de los certificados estarán revocados):

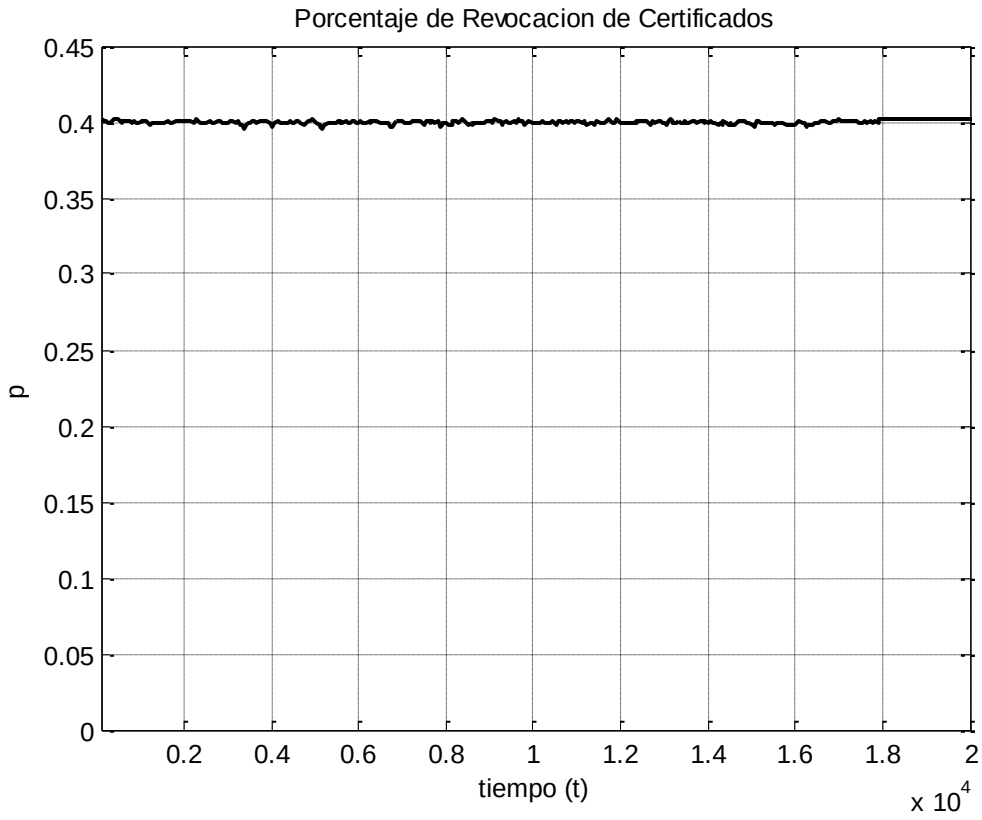


Figura 5.7. Evolución del Porcentaje de Revocación de Certifcados en un entorno sin interrupciones.

Por último, en la Figura 5.8 podemos analizar la evolución de los distintos conjuntos de certificados involucrados en el proceso de revocación. Como podemos apreciar, inicialmente hay un pequeño salto, que corresponde al periodo del transitorio durante el cual se esta "*cargando*" el sistema y se establecen las condiciones iniciales de la simulación, las cuales corresponderán con los parámetros especificados en el fichero de configuración.

Luego de este periodo transitorio, los certificados comienzan a expirar y las solicitudes de revocación son enviadas aleatoriamente.

- El **Número de Certificados No Expirados *C(t)*** se estabiliza alrededor del valor de 1000 certificados.

- Del mismo modo, el **Número de Certificados Revocados No Expirados *V(t)*** se estabiliza alrededor de 400 certificados (*V = pC = 0.4 x 1000 = 400*).

- La Evolución de los **Certificados Revocados Desconocidos *U(t)***, en este caso, al ser un entorno donde las interrupciones son casi inexistentes, es un número muy bajo comparado con el número de Certificados Emitidos No Expirados *C(t)*. Como es esperado, después de cada emisión de una nueva CRL los usuarios conocerán inequívocamente todos los Certificados Revocados del Sistema.

- Analizando la Evolución del **Número de Certificados Operativos *O(t)***, podemos ver que este conjunto crece y decrece debido a las variaciones de *U(t)* y *C(t)*. Sin embargo, como *U(t)* es relativamente muy pequeño comparado con *C(t)* (debido a que *C(t)* se mantiene casi constante y U(t) difícilmente varía), el número de Certificados Operativos no varía visiblemente.

- Por tal motivo el **Número de Certificados Buenos *G(t)*** y el Numero de Certificados Operativos *O(t)* es casi el mismo. Los Certificados Buenos *G(t)* se calculan como la diferencia entre los Certificados Emitidos No Expirados y los Certificados Revocados No Expirados (*G = C – V = 1000 – 400 = 600 certificados*).

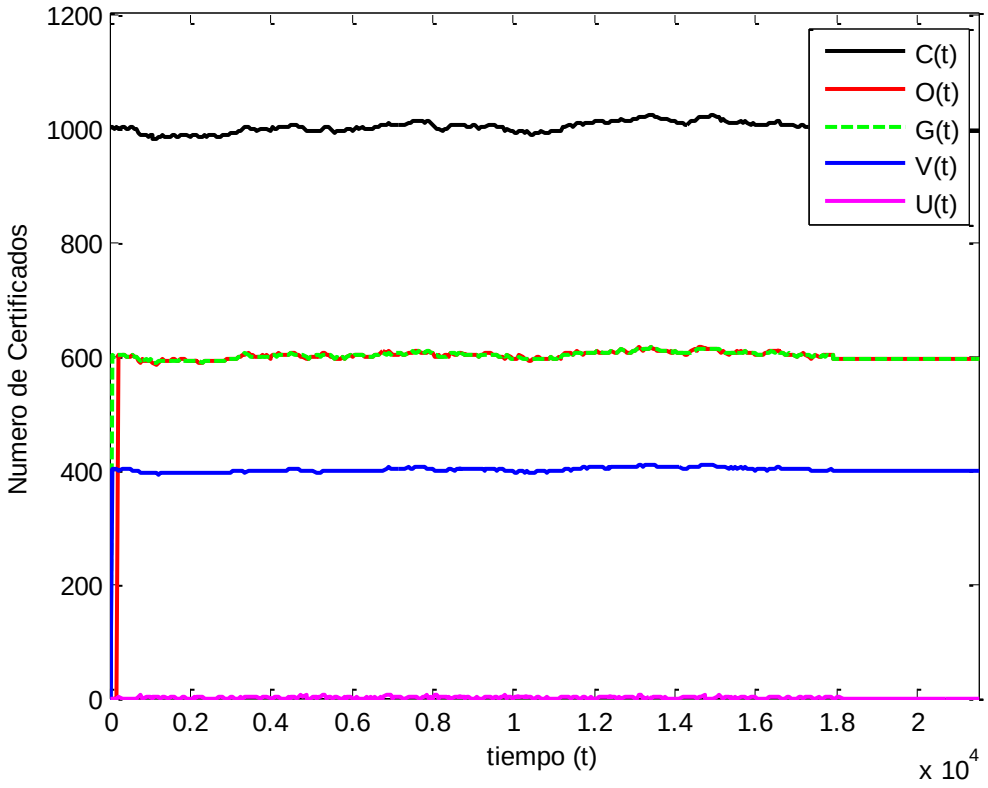


Figura 5.8. Conjuntos de Certificados a lo largo del tiempo en un entorno sin interrupciones.

Una vez analizada la evolución de los diferentes conjuntos de certificados, podemos calcular el riesgo a partir de la relación:

$$riesgo = \frac{E[U(t)]}{E[O(t)]} = \frac{U}{O}$$

La Figura 5.9 muestra la evolución del riesgo a lo largo de toda la simulación. Con esta configuración, como casi no sufrimos desconexiones, el riesgo de usar una CRL almacenada en la caché del usuario es bastante bajo (alrededor del 0.08% de riesgo). Como es esperado, el valor del riesgo es cero cuando recibimos una CRL actualizada, y a partir de allí empieza a crecer. En este caso el riesgo no alcanza el valor teórico máximo, que es igual al Porcentaje de Certificados Revocados en el Sistema (*p = 40%*). Esto ocurre debido al hecho

de que el Número de Certificados Revocados Desconocidos nunca alcanza al Número Total de Certificados Revocados. Hay principalmente dos razones para esto:

- Los usuarios casi siempre están conectados, por lo que pueden obtener una CRL actualizada inmediatamente es generada en el servidor.
- El intervalo de emisión de una nueva CRL es muy pequeño comparado con el tiempo de expiración de un certificado revocado (teniendo en cuenta que el tiempo entre ocurrencias de una expiración aleatoria de un certificado es una variable aleatoria exponencial con una media de 25 minutos).

Debido a esto los usuarios siempre conocerán los certificados revocados que aún no han expirado.

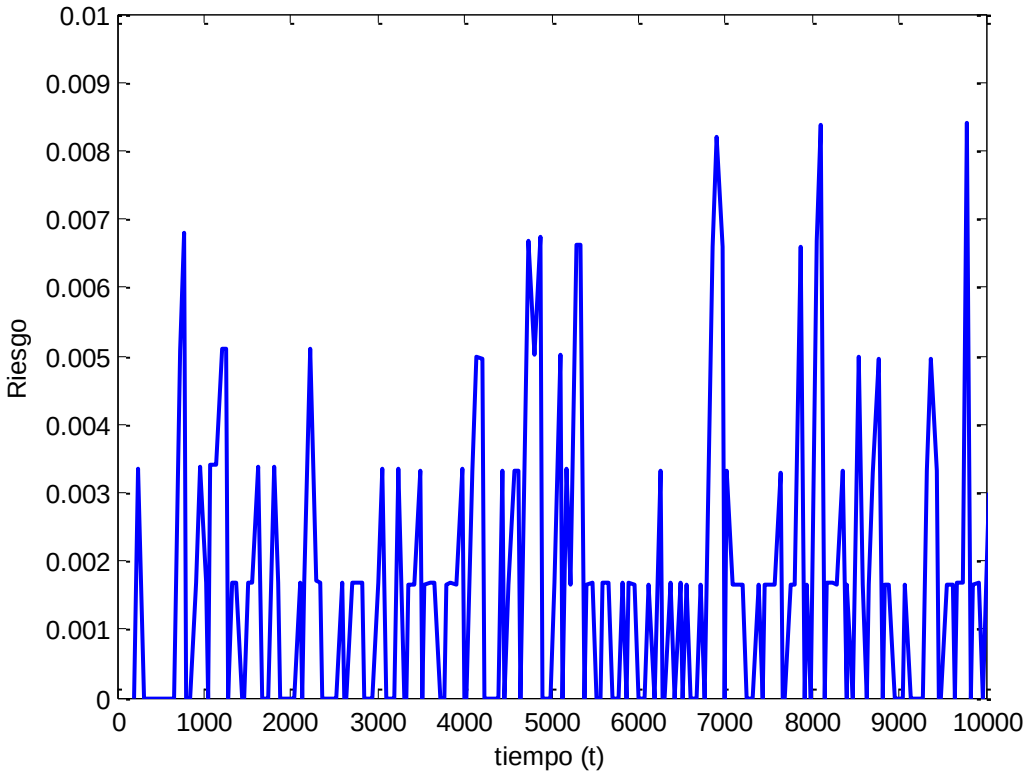


Figura 5.9. Evolución del Riesgo en un entorno sin interrupciones.

5.5 Evaluación del Riesgo en la Emisión periódica de Información de Estado de Certificados en un entorno de conectividad baja

En este escenario, asumiremos que los nodos de la red son altamente disruptivos, es decir, que sufren constantemente de pérdidas de conectividad.

Esta vez configuraremos los parámetros de la siguiente manera:

- El Número de Certificados en el Sistema será 3000.
- El Porcentaje de Revocación *p* será de 10%.
- Tiempo entre llegada de eventos: 6 segundos.

$$revocationsRate = \frac{6}{1 - 0.1} = 6.66\ seg$$

$$expirationsRate = \frac{6}{0.1} = 60\ seg$$

- Periodo de validez de CRL: 480 minutos.
- CRL Sobre-emitida: No. Se emitirá 1 sola CRL en cada periodo de validez.
- Tiempo de Simulación: 28800 minutos
- Probabilidad de Conexión del Cliente: $P_C = 0.15$, $P_D = 0.85$

Como vemos, este escenario presentará una conectividad muy baja (tendremos 85% de probabilidades de que el cliente esté desconectado) y además estará lleno de eventos de expiraciones de certificados, ya que la tasa de expiraciones es bastante reducida (60 seg).

Los resultados obtenidos son los siguientes:

El Número de Certificados No Expirados en el Sistema se mantiene constante alrededor de los 3000 certificados:

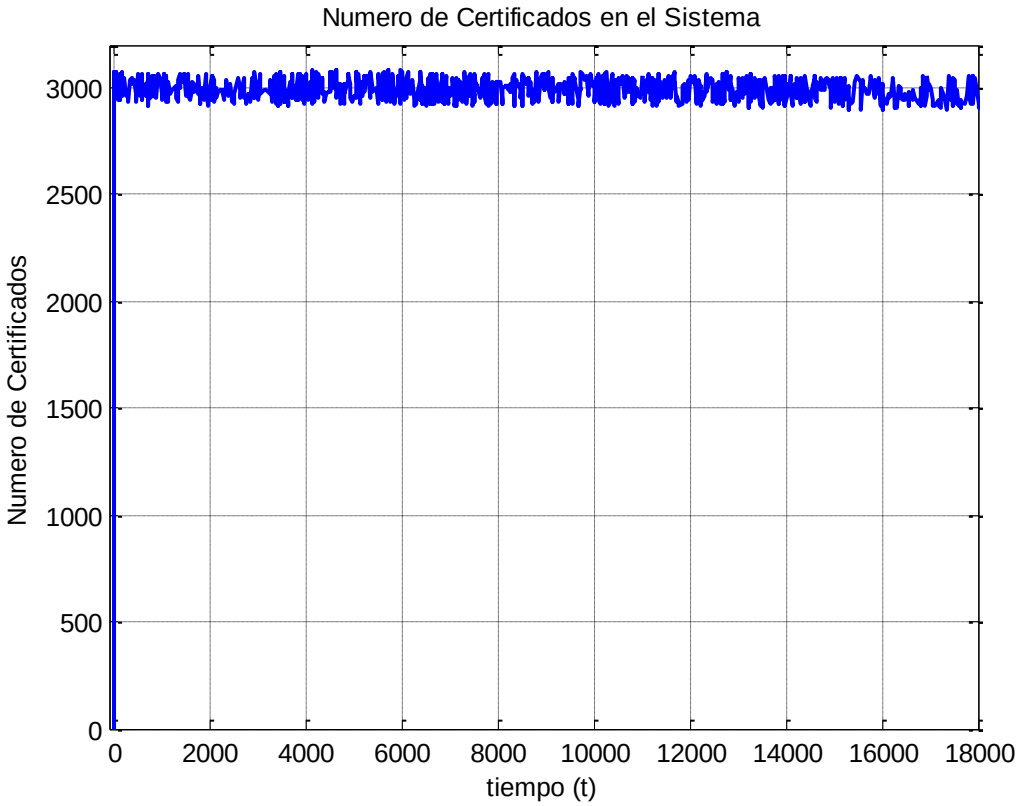


Figura 5.10. Evolución del Número de Certificados Emitidos No Expirados en el Sistema en un entorno de conectividad baja o limitada.

Del mismo modo, el Número de Certificados Revocados No Expirados en el Sistema se mantiene constante alrededor de los 300 certificados:

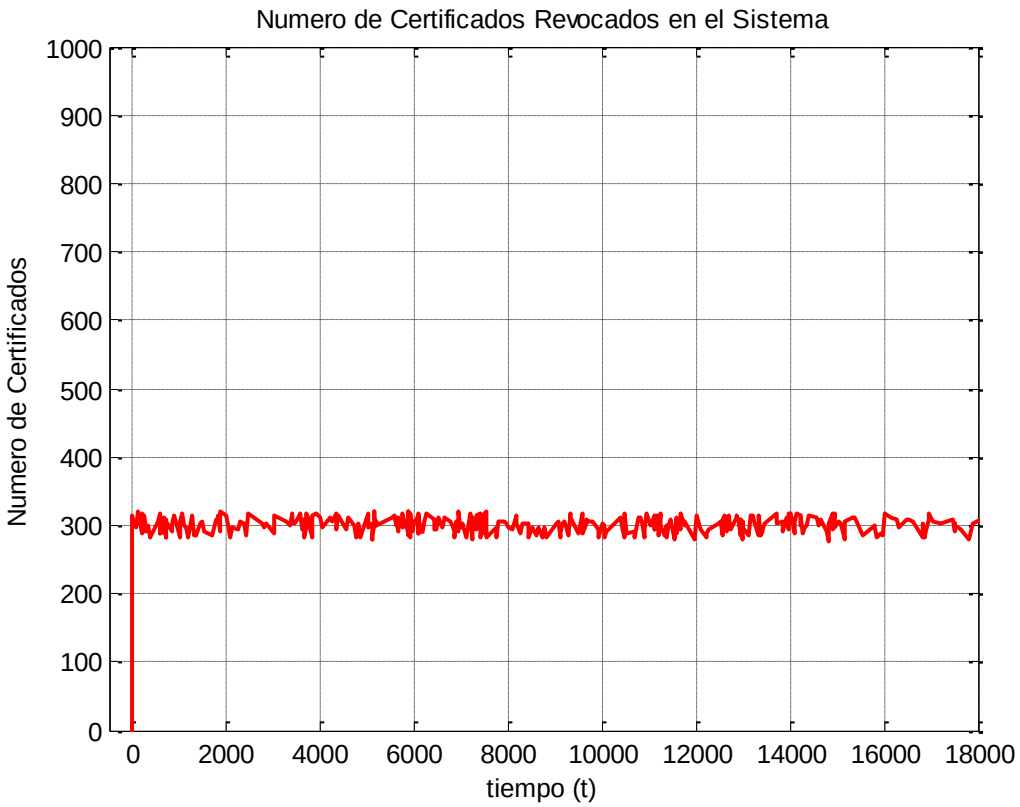


Figura 5.11. Evolución del Número de Certificados Revocados en el Sistema en unentorno de conectividad baja o limitada.

Vemos que el valor del porcentaje de revocación es estable, y se mantiene, como es esperado, sobre el 10%.

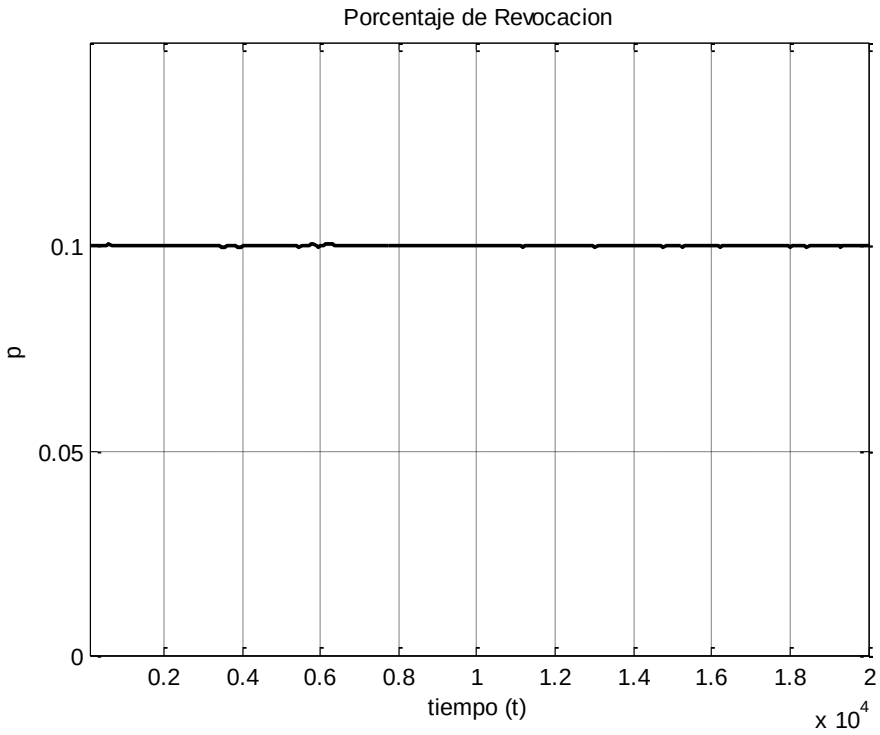


Figura 5.12. Evolución del Porcentaje de Revocación de Certificados en un entorno de conectividad baja o limitada.

La Figura 5.13 muestra la Evolución de los Conjuntos de Certificados durante toda la simulación. La principal diferencia apreciable en dicha figura, con respecto a la del escenario anterior, corresponde al Número de Certificados Revocados Desconocidos *U(t)* y al Número de Certificados Operativos *O(t).*

En esta configuración, al haber grandes periodos de desconexión y muchas expiraciones, el Número de Ceritificados Desconocidos *U(t)* crece a buen ritmo.

En este sentido, ocurre que en algún momento *U(t)* alcanza a *V(t)*, es decir, todos los Certificados Revocados No Expirados son desconocidos para el usuario.

Es más, debido a la baja conectividad, *U(t)* solo decrece en unas pocas ocasiones, esto es, cuando se recupera la conectividad y el usuario descarga una nueva CRL.

Con respecto a los Certificados Operativos, en este caso estos crecen mucho más con respecto a la simulación previa, alcanzando inclusive a *C(t)*. Este crecimiento se debe principalmente al aumento de Certificados Desconocidos *K(t)*. En este sentido, cada cierto intervalo de tiempo ocurrido desde la descarga de una CRL y estando esta sin haber sido actualizada, los usuarios no conocerán realmente cuales certificados están revocados y cuales no, de modo que asumirán cualquier certificado como Operativo, es decir, cualquier Certificado Emitido No Expirado estará para él en estado operativo.

Esto es debido a que de acuerdo a su CRL desactualizada, los Certificados que el cree que están Revocados ya habrán expirado en su gran mayoría, y la mayoría de los certificados que forman parte del conjunto *C(t)* habrán sido emitidos posteriormente a la descarga de su última CRL. Por tanto, ninguno de estos "nuevos" certificados estará en su lista, y por ello verá todos los certificados como si fueran Certificados Operativos.

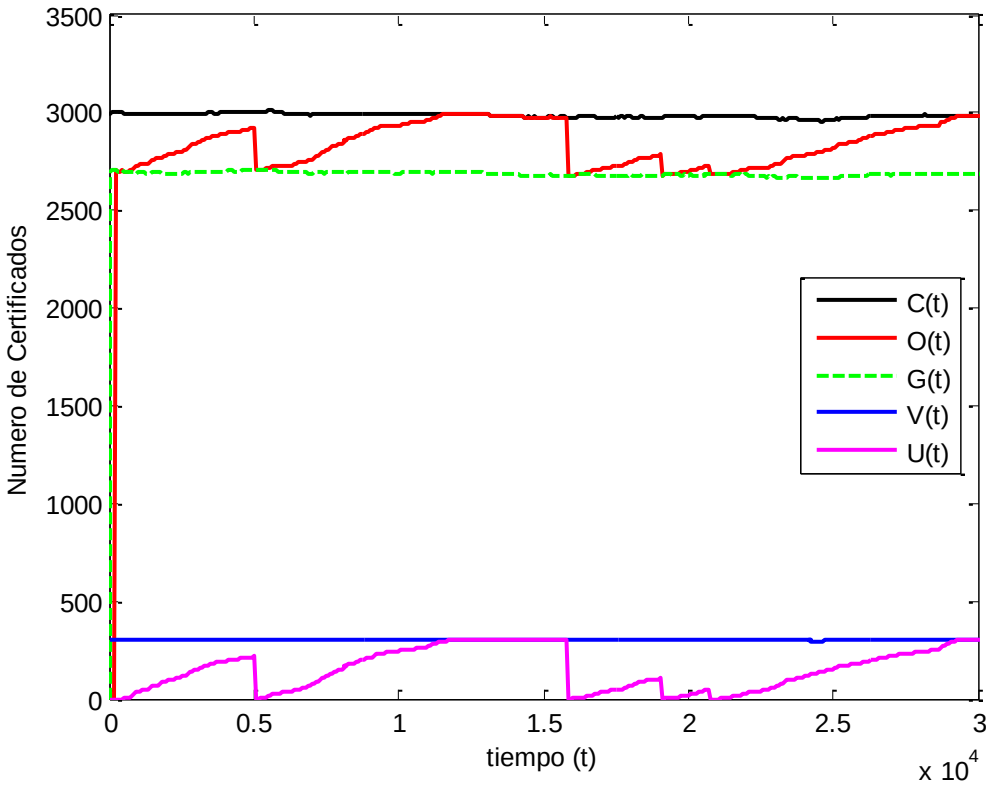


Figura 5.13. Conjunto de Certificados a lo largo del tiempo en entornos con conectividad baja o limitada.

Si comparamos la Evolución del Riesgo en este escenario con respecto al anterior, podemos notar que en este caso el riesgo alcanza el Porcentaje Total de Certificados Revocados (*p = 10% = 0.1*). Esto ocurre únicamente durante periodos largos de desconexión, cuando todos los certificados que están revocados de acuerdo a su última CRL descargada ya han expirado realmente.

En este punto, los estados de todos los certificados son desconocidos, por ello la probabilidad de usar un certificado revocado será igual a la probabilidad global de que un certificado sea revocado (p). Una vez que el usuario recupera la conexión, podrá descargar la CRL actual y actualizar su estado.

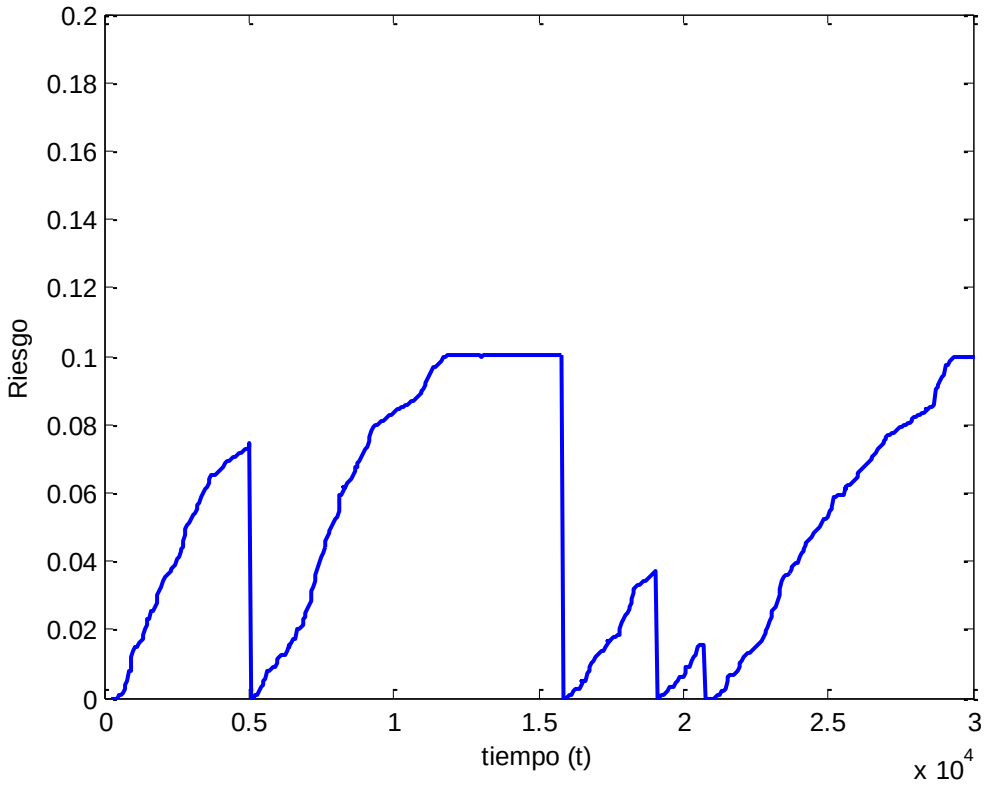


Figura 5.14. Evolución del Riesgo en entornos de conectividad baja o limitada.

Capítulo 6

CONCLUSIONES Y LÍNEAS FUTURAS

Después de la implementación del presente proyecto, y de las simulaciones realizadas para la comprobación de su correcto funcionamiento, lo primero que afirmamos es que esta Plataforma funciona correctamente como un Servidor de Revocaciones Digitales.

A lo largo del desarrollo de esta aplicación, se han ido simulando y probando diferentes casos de uso para poder refinar el comportamiento del proyecto, y después de mucho trabajo, se ha conseguido tener una aplicación que funcione correctamente, tanto en modo Estandar comportándose como un Servidor Real y respondiendo a las solicitudes que le puedan enviar, así como en el modo Test, donde la generación automática de procesos aleatorios funciona eficazmente.

Un detalle que cabe resaltar es el gran acierto de haber elegido el lenguaje de programación Python para el desarrollo de la Plataforma. Python realmente es un lenguaje muy fácil de usar, muy sencillo de aprender y cuya gestión de errores y mensajes de fallos es bastante bueno. Además, una importante ventaja que ofrece es que existen una gran variedad de librerías que dan soporte para distintas funcionalidades que han sido vitales en el desarrollo del proyecto: Manejo de cadenas de bits, manejo de elementos ASN.1 y soporte integrado con la librería OpenSSL para la gestión, edición, firma y verificación de Certificados Digitales X.509. Por tanto,

podemos decir que Python es un lenguaje de programación bastante potente y realmente recomendado para aplicaciones que requieran soportar una gran cantidad de procesamiento de datos.

Otra conclusión que podemos obtener es que el haber implementado una Plataforma de Revocación de Certificados Digitales, detallando además la lógica de funcionamiento, así como todos los módulos que la componen y las tareas que cada uno de ellos realiza, se ha podido comprender la Importancia de los Certificados Digitales, del Proceso de Revocación y Mantenimiento de Listas de Certificados Revocados, ya que sin estos elementos sería prácticamente imposible poder llevar a cabo cualquier tipo de transacción electrónica sin tener la certeza de que ha sido segura y que se ha llevado a cabo correctamente, así como evitar cualquier posibilidad de fraude.

Se ha demostrado la eficiencia del uso de CRLs sobre-emitidas: emitiendo más de una CRL en cada periodo de validez permite evitar que el Servidor pase intervalos de tiempo donde se encuentre prácticamente inactivo, o intervalos de tiempo donde este muy saturado debido a la gran cantidad de solicitudes concurrentes de descarga de CRL recibida por parte de los usuarios.

Además, después de haber estudiado el Riesgo y el Mecanismo de Emisión de CSI con Riesgo Constante, cabe resaltar que es de vital importancia en la comunicación de estados de certificados en VANETs. Realmente este mecanismo proporciona un soporte adicional para poder trabajar en estos entornos (VANETs o DTNs), los cuales comúnmente están plagados de desconexiones y pérdidas conectividad. Con este mecanismo se permite dar una ayuda para poder interactuar (siempre asumiendo un cierto *riesgo*) con Certificados de los cuales no estamos seguros si son completamente confiables o no. Uno de los objetivos de este proyecto en cuanto a este punto era desarrollar un modo para poder evaluar el riesgo que existe realmente desde el punto de vista de un usuario, es decir, el riesgo calculado con todos los conjuntos de certificados reales y sus valores actualizados en cada momento. Esto se consiguió implementando una interfaz adicional que a manera de una API, permite al cliente poder consultar las estadísticas necesarias para el cálculo del Riesgo. Por tanto podemos decir que este obejtivo ha sido cumplimentado.

Otra conclusión que podemos aportar es que si bien, dentro del desarrollo de este proyecto se ha dicho que implementando algoritmos de movilidad se quería simular estados de conexión y de desconexión de clientes para poder servir como escenario

para la evaluación del mecanismo implementado para cálculo del riesgo, este método no es el más adecuado para hacerlo. Al ser estos modelos un tanto aleatorios y continuos, realmente depende del nivel de esta aleatoriedad poder determinar si el cliente cae dentro de la cobertura (conexión) o fuera de ella (desconexión). Las simulaciones desarrolladas con este mecanismo realmente no han sido satisfactorias. Por tanto, a pesar de que en el código fuente mantiene esta implementación, así como se mantiene en el fichero de configuración (parámetro *disconnectionsAlgorithm*), no se recomienda su uso para la evaluación del mecanismo del riesgo. Como este proyecto está enfocado en dicha evaluación, al ser las simulaciones poco exitosas, se han descartado de incluirse en el apartado de Simulaciones.

Si bien la plataforma está terminada y ya es completamente funcional, aún quedan algunos puntos pendientes que pueden contribuir a mejorar su funcionamiento y algunos detalles que harán falta revisar y evaluar. Entre ellos tenemos principalmente dos asuntos que quedan pendientes a futuros desarrollos:

- Generar una CRL que puede ser utilizada con algún navegador que actualmente esté en uso, por ejemplo Mozilla Firefox. La CRL generada por esta plataforma tiene un formato correcto, ya que sigue la notación ASN.1 de las CRLs X.509 especificada en el RFC correspondiente. Utilizando la librería OpenSSL se comprueba su autenticidad y validez, sin embargo al intentar importarlo a un navegador, no se ha conseguido tener éxito.

- Integrar la Plataforma de Revocación de Certificados Digitales con una Autoridad de Certificación. Con esto, tendríamos completamente la información de todos los certificados: Podríamos emitir certificados, conocer la fecha de expiración de los Certificados, renovar un certificado luego de que haya expirado o haya sido revocado, resetear el par de claves, etc. Es decir, unir las piezas y crear verdaderamente una Infraestructura de Clave Pública.

APÉNDICES

Apéndice A. Descripción ASN.1 de los Elementos a utilizar

A.1. Descripción ASN.1 de un Certificado X.509

```
Certificate  ::=  SEQUENCE  {
     tbsCertificate       TBSCertificate,
     signatureAlgorithm   AlgorithmIdentifier,
     signature            BIT STRING  }


TBSCertificate  ::=  SEQUENCE  {
     version         [0]  Version DEFAULT v1,
     serialNumber         CertificateSerialNumber,
     signature            AlgorithmIdentifier,
     issuer               Name,
     validity             Validity,
     subject              Name,
     subjectPublicKeyInfo SubjectPublicKeyInfo,
     issuerUniqueID  [1]  IMPLICIT UniqueIdentifier OPTIONAL,
                          -- If present, version MUST be v2 or v3
     subjectUniqueID [2]  IMPLICIT UniqueIdentifier OPTIONAL,
                          -- If present, version MUST be v2 or v3
     extensions      [3]  Extensions OPTIONAL
                          -- If present, version MUST be v3 --  }


Version  ::=  INTEGER  {  v1(0), v2(1), v3(2)  }


CertificateSerialNumber  ::=  INTEGER


Validity ::= SEQUENCE {
     notBefore      Time,
     notAfter       Time  }


Time ::= CHOICE {
     utcTime        UTCTime,
     generalTime    GeneralizedTime }


UniqueIdentifier  ::=  BIT STRING
```

```
SubjectPublicKeyInfo  ::=  SEQUENCE  {
     algorithm            AlgorithmIdentifier,
     subjectPublicKey     BIT STRING  }


Extensions  ::=  SEQUENCE SIZE (1..MAX) OF Extension

Extension  ::=  SEQUENCE  {
     extnID      OBJECT IDENTIFIER,
     critical    BOOLEAN DEFAULT FALSE,
     extnValue   OCTET STRING  }


AlgorithmIdentifier  ::=  SEQUENCE  {
     algorithm               OBJECT IDENTIFIER,
     parameters              ANY DEFINED BY algorithm OPTIONAL  }
                                -- contains a value of the type
                                -- registered for use with the
                                -- algorithm object identifier value
```

A.2. Descripción ASN.1 de una CRL

```
CRL DEFINITIONS EXPLICIT TAGS::=

IMPORTS

-- PKIX Certificate Extensions

      ReasonFlags, DistributionPointName, GeneralNames, CRLNumber,
      BaseCRLNumber, IssuingDistributionPoint, CRLReason,
      CertificateIssuer, HoldInstructionCode, holdInstruction,
      InvalidityDate, id-ce
      FROM PKIX1Implicit88 {iso(1) identified-organization(3)
            dod(6) internet(1) security(5) mechanisms(5) pkix(7)
            id-mod(0) id-pkix1-implicit-88(2)}

      Version, Time, Name, CertificateSerialNumber, Extensions,
      CertificateList, TBSCertList, AlgorithmIdentifier,
      DirectoryString
      FROM PKIX1Explicit88 {iso(1) identified-organization(3)
            dod(6) internet(1) security(5) mechanisms(5) pkix(7)
            id-mod(0) id-pkix1-explicit-88(1)};



DeltaCRLIndicator ::= BaseCRLNumber


reasonCode ::= { CRLReason }


ID     ::=    OBJECT IDENTIFIER

sha-1WithRSAEncryption OBJECT IDENTIFIER  ::=  {
            iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1)
            pkcs-1(1) 5  }


md2WithRSAEncryption OBJECT IDENTIFIER  ::=  {
            iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1)
            pkcs-1(1) 2  }


md5WithRSAEncryption OBJECT IDENTIFIER  ::=  {
            iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1)
            pkcs-1(1) 4  }


id-dsa-with-sha1 ID  ::=  {
                  iso(1) member-body(2) us(840) x9-57 (10040)
                  x9cm(4) 3 }


CertificateList  ::=  SEQUENCE  {
     tbsCertList          TBSCertList,
     signatureAlgorithm   AlgorithmIdentifier,
     signature            BIT STRING  }
```

```
TBSCertList  ::=  SEQUENCE  {
     version                 Version OPTIONAL,
                                  -- if present, MUST be v2
     signature               AlgorithmIdentifier,
     issuer                  Name,
     thisUpdate              Time,
     nextUpdate              Time OPTIONAL,
     revokedCertificates     SEQUENCE OF SEQUENCE  {
          userCertificate         CertificateSerialNumber,
          revocationDate          Time,
          crlEntryExtensions      Extensions OPTIONAL
                                        -- if present, MUST be v2
                               }  OPTIONAL,
     crlExtensions           [0] Extensions OPTIONAL }
                                        -- if present, MUST be v2
```

A.3. Definición ASN.1 del Protocolo SCRP

```
SCRP DEFINITIONS EXPLICIT TAGS::=

IMPORTS


-- PKIX Certificate Extensions
      CRLReason
      FROM PKIX1Implicit88 {iso(1) identified-organization(3)
            dod(6) internet(1) security(5) mechanisms(5) pkix(7)
            id-mod(0) id-pkix1-implicit-88(2)}

      -- AlgorithmIdentifier,
      Certificate
      FROM PKIX1Explicit88 {iso(1) identified-organization(3)
            dod(6) internet(1) security(5) mechanisms(5) pkix(7)
            id-mod(0) id-pkix1-explicit-88(1)};



AlgorithmIdentifier  ::=  SEQUENCE  {
     algorithm               OBJECT IDENTIFIER,
     parameters              NULL OPTIONAL
}


SCRPRequest     ::=     SEQUENCE {
    tbsRequest          Request,
    signature           Signature }


Signature       ::=     SEQUENCE {
    signatureAlgorithm  AlgorithmIdentifier,
    signature           BIT STRING,
    certs               [0] EXPLICIT SEQUENCE OF Certificate OPTIONAL
}


Request ::=     SEQUENCE {
    certToRev           Certificate,
    issuerCert          Certificate,
    revocationTime      GeneralizedTime,
    revocationReason    [0] EXPLICIT CRLReason OPTIONAL }


SCRPResponse ::= SEQUENCE {
    tbsResponse         Response,
    signature           Signature }


Response ::= SEQUENCE {
    certToRev           Certificate,
    result              Result }
```

```
Result ::= ENUMERATED {
    successful            (0),      --ok
    alreadyRevoked        (1),      --cert was already revoked
    internalError         (2),      --bad news
    sigRequired           (3),      --Must sign the request
    unauthorized          (4)       --Request unauthorized
}
```

Apéndice B. Manual de Usuario de la Aplicación

Antes de iniciar la ejecución del Servidor de Revocaciones, debemos verificar algunas características que permitirán el funcionamiento correcto de la Plataforma de Revocación de Certificados Digitales.

Los elementos a tener en cuenta son esencialmente dos: Base de Datos y Lenguaje de Programación Python.

B.1 Instalación y Configuración de la Base de Datos

Primero debemos comprobar que la Base de Datos esté instalada y configurada correctamente. Si es la primera vez que ejecutamos la aplicación, deberemos realizar los siguiente pasos para preparar la Base de Datos de la plataforma:

La BD utilizada es MySQL, y debemos verificar inicialmente si tenemos MySQL instalado. En un sistema Linux podemos ejecutar la instrucción:

```
# mysql –u root –p
```

En el caso que dicho comando falle, significa que hará falta instalar MySQL en la máquina. Para ello, podemos hacerlo de la siguiente manera:

```
# sudo apt-get update
# sudo apt-get install mysql-server
# mysql –u root –p
```

Con este último comando accedemos a la consola de MySQL.

Lo siguiente será crear la BD que usaremos para almacenar los datos de la plataforma de revocación, así como el usuario que utilizaremos para conectarnos a ella.

Por ejemplo, si queremos crear la Base de datos con nombre “**cervantes**”, a la cual accederemos con el usuario **“user”** y contraseña **“user1234”** desde la misma máquina donde está instalado el servidor, debemos ejecutar:

```
#  mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands END with ; or \g.
Your MySQL connection id is 9
Server version: 5.0.51a-3ubuntu5.1 (Ubuntu)
Type 'help;' or '\h' for HELP. Type '\c' to clear the buffer.

mysql> CREATE DATABASE cervantes;
 Query OK, 1 row affected (0.22 sec)
mysql> GRANT ALL PRIVILEGES ON cervantes.* TO 'user'@'localhost'
IDENTIFIED BY 'user1234' WITH GRANT OPTION;
Query OK, 0 rows affected (0.19 sec)
mysql> FLUSH PRIVILEGES;
```

Una vez creado el usuario y la BD, ejecutaremos el script para la creación de tablas de la BD que emplea la plataforma implementada.

Dentro de la carpeta de la aplicación, en la siguiente ruta: ***/sql/mysql_script.sql*** tenemos el script SQL de creación de tablas. Podemos ejecutarlo desde una terminal de Linux accediendo a dicha ruta y ejecutando el siguiente comando:

```
#mysql –u root –p [root_password] [database_name] > mysql_script.sql
```

Con esto habremos completado la configuración de la Base de Datos.

B.2 Instalación y configuración de Python

La aplicación está desarrollada en el lenguaje de programación Python. Por tanto se requiere tenerlo instalado antes de ejecutar el servidor. La plataforma ha sido desarrollada en Python 2.6.5. Se ha comprobado que funciona sin ningún problema con las versiones Python 2.4, 2.5 y 2.7. Sin embargo, **no se podrá ejecutar la aplicación con Python 3.**

Si no lo tenemos instalado, hará falta descargarlo de su página web: http://www.python.org. Se recomienda descargar la versión 2.6.5. Una vez descargada, seguir las instrucciones allí indicadas para su instalación.

Adicionalmente hacen falta instalar algunas librerías que son utilizadas en la aplicación. Podemos instalarlas de la siguiente manera:

```
# sudo apt-get install python-openssl, python-egenix-mxdatetime, python-m2crypto,
python-mysqldb, python-tk, python-pyasn1, python-crypto
```

Ejecutando el Servidor de la Plataforma

Para ejecutar la aplicación, primero debemos revisar el fichero de configuración del Servidor. Los principales parámetros que debemos verificar para el correcto funcionamiento son:

- **Mode:** modo de ejecución del servidor.
  mode=0 (Modo Estándar), mode=1 (Modo Test)

- Parámetros para la configuración de la conexión de la base de datos:
  - *user*: Nombre del usuario con el que accedemos a la BD.
  - *password*: Contraseña de dicho usuario.
  - *host*: Dirección IP de ubicación de la BD. Será 'localhost' si está en la misma máquina donde se ejecutará el servidor.
  - *port*: Puerto configurado para la conexión. Por defecto, en MySQL es el 3306.
  - *db*: Nombre de la BD

Los demás parámetros configuran el comportamiento de la simulación, los cuales ya han sido explicados en el Capítulo 3 de este proyecto, correspondiente a la Implementación de la Plataforma.

Una vez que completemos los datos del fichero de configuración, procederemos a ejecutar el Servidor. El script de ejecución del servidor se encuentra en: ***/src/server/runServer.py***
Debemos acceder a dicha ubicación y ejecutarlo de la siguiente manera:

```
#python runServer.py
```

Cuando la aplicación está en uso, podremos observar una serie de trazas en la consola de ejecución que nos informarán acerca los procesos de Inicialización del Servidor que se están ejecutando, así como los errores que puedan ocurrir.

```
ronaldo@ronaldo-PC:~/Escritorio/PFC_v0.5.1/src/server$ python runServer.py
Initializing Revocation Servers Manager ...
----> General  --  mode = 0
----> General  --  numberOfCertificates = 1000
----> Database  --  user = user
----> Database  --  password = user1234
----> Database  --  host = localhost
----> Database  --  port = 3306
----> Database  --  db = cervantes
----> Database  --  clearDatabaseContents = TRUE
----> Stage  --  height = 10
----> Stage  --  width = 15
----> Stage  --  x = 5
----> Stage  --  y = 5
----> Stage  --  radius = 5
----> Test  --  revocationPerc = 0.4
----> Test  --  eventsRate = 10
----> Test  --  testDuration = 18000
----> Test  --  revocationsLog = ./log/SCRP_test.log
----> Test  --  issuerCertificate = ./certs/public/cervantes_issuer.X509.Certificate.bin
----> Test  --  disconnectionsAlgorithm = 2
----> SCRP  --  type = SCRP
----> SCRP  --  class = SCRPPortThread
----> SCRP  --  name = SCRP01
----> SCRP  --  revokerCertificate = ./certs/public/revoker.X509.Certificate.bin
----> SCRP  --  responderCertificate = ./certs/public/cervantes_responder.X509.Certificate.bin
----> SCRP  --  responderPrivateKey = ./certs/private/cervantes_responder.PKCS8.EncryptedPrivateKeyInfo.pem
----> SCRP  --  serverLog = ./log/SCRP01.log
----> SCRP  --  port = 5001
----> SCRP  --  cleanExpiredCertsThread = 60
----> CRL  --  type = CRL
----> CRL  --  class = CRLPortThread
----> CRL  --  name = CRL01
----> CRL  --  validityPeriod = 240
----> CRL  --  overissuedFactor = 4
----> CRL  --  certificatePath = ./certs/public/revoker.X509.Certificate.bin
----> CRL  --  privateKeyPath = ./certs/private/cervantes_responder.PKCS8.EncryptedPrivateKeyInfo.pem
----> CRL  --  port = 6001
----> CRL  --  serverLog = ./log/CRL01.log
Properties loaded from : ./etc/server_conf.txt
Start mode : 0
Initialize DB ...
Transaction instance has no attribute 'cur'
Standard mode : Database cleaned! (clearDatabaseContents = TRUE)
Creating threads for port Listeners ...
Creating server SCRP --> SCRP01
[Sun Nov 14 02:02:23 2010] Creating SCRP Server --> SCRPPortThread::__init__

[Sun Nov 14 02:02:23 2010] Start Mode : 0

[Sun Nov 14 02:02:23 2010] Database connection OK!

SCRP -- STANDARD MODE
[Sun Nov 14 02:02:23 2010] Initializing CleanExpiredCertsThread ...

Creating server CRL --> CRL01
Starting Server 0 ...
OPENING SERVER SOCKET :: LISTENING SCRP REQUESTS TO PORT 5001
Starting CleanExpiredCerts Thread!
Starting Server 1 ...
```

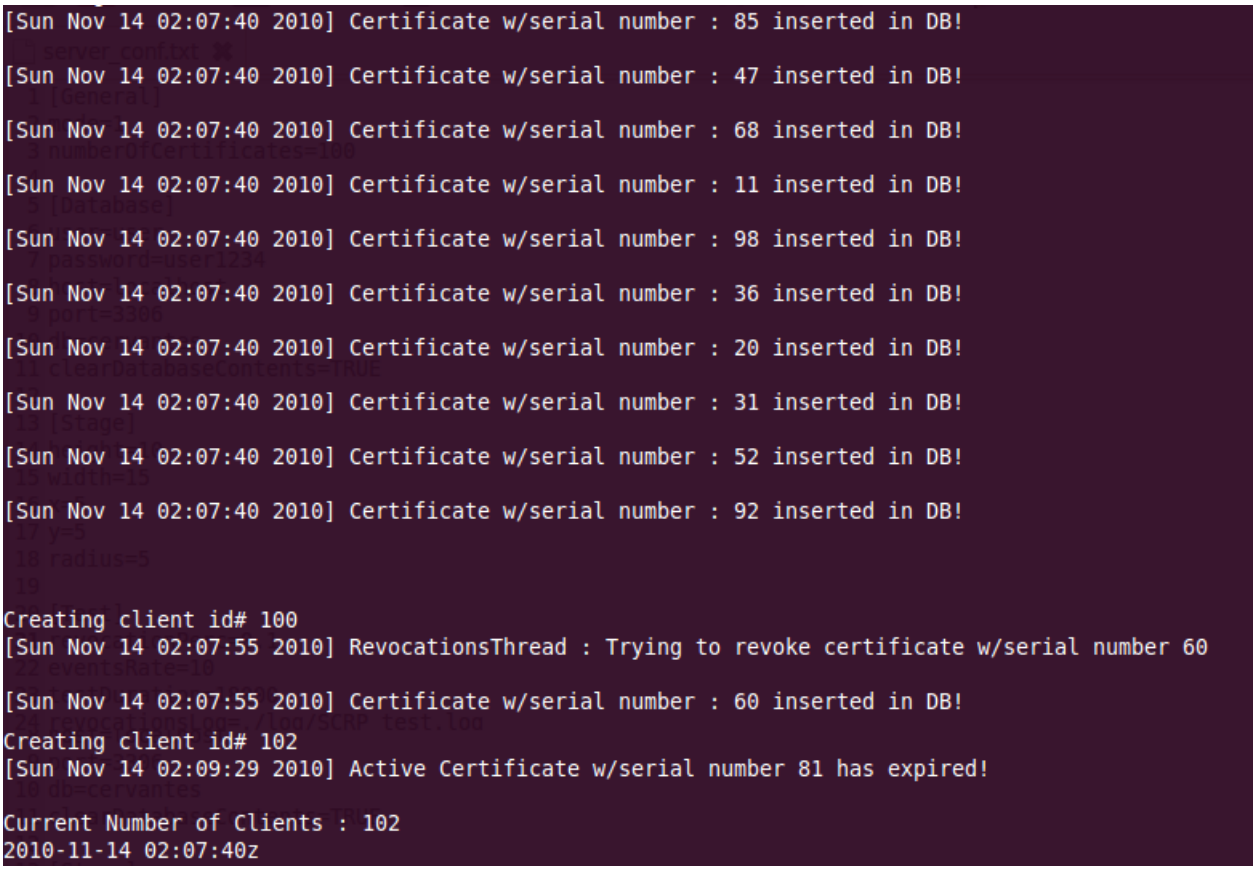
Trazas de salida al iniciar el Servidor

En el modo STANDARD (mode=0) el servidor se inicializará y se pondrá en estado de escucha, esperando recibir peticiones de clientes, tanto solicitando la descarga de la CRL, como solicitando la revocación de un certificado.

```
[Sun Nov 14 02:03:55 2010] Recibo conexion de 127.0.0.1:51730

[Sun Nov 14 02:03:55 2010] Received :: 1447 BYTES

[Sun Nov 14 02:03:55 2010] Certificate to Revoke :

[Sun Nov 14 02:03:55 2010] Certificate:
    Data:
        Version: 1 (0x0)
        Serial Number: 29 (0x1d)
        Signature Algorithm: sha1WithRSAEncryption
        Issuer: C=ES, ST=Barcelona, L=Barcelona, O=Retevision, OU=Provision, CN=Isidoro Bernabe
        Validity
            Not Before: Apr 22 06:15:00 2002 GMT
            Not After : Jun  8 21:14:00 2002 GMT
        Subject: C=ES, ST=Barcelona, L=Barcelona, O=Retevision, OU=Provision, CN=Isidoro Bernabe
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
            RSA Public Key: (1024 bit)
                Modulus (1024 bit):
                    00:b9:09:bf:a4:29:07:89:94:55:b6:42:8e:4f:4b:
                    76:f4:f0:0b:4f:07:d8:91:83:3f:0b:eb:fb:08:97:
                    cd:c4:d3:48:ca:c6:82:3c:3c:e8:6c:fa:44:2f:54:
                    c6:4d:2f:26:fb:90:60:dc:9b:c5:9a:10:f8:ec:32:
                    fd:bf:47:2b:0b:f6:7e:d1:a6:4b:41:58:d1:c9:d8:
                    e2:5f:5a:21:63:6b:14:28:ea:81:bb:76:e2:a3:d5:
                    e2:b0:f8:67:33:14:0b:ea:78:5b:30:1b:bf:14:83:
                    7b:37:fb:3d:c1:63:75:32:08:e2:bc:9c:34:cd:80:
                    11:13:03:3e:01:ac:a1:0b
                Exponent: 65537 (0x10001)
    Signature Algorithm: sha1WithRSAEncryption
        40:04:22:6f:46:6a:10:1b:53:4f:38:20:f5:f9:26:16:f4:1d:
        c2:e4:7f:72:a3:df:23:5a:d4:a3:45:55:43:dd:67:16:da:f2:
        37:b3:0c:f8:d7:15:0c:a6:b0:83:7a:95:5f:5f:6b:56:93:69:
        7f:4c:74:02:3f:24:bb:a2:c8:7b:d5:52:da:26:e4:64:cd:4c:
        b4:17:00:15:1a:76:ef:1a:c9:17:87:a0:52:4f:ae:65:96:28:
        cf:86:b7:9a:64:2e:da:9a:b1:62:11:7c:20:17:74:bf:92:76:
        d5:66


[Sun Nov 14 02:03:55 2010] Issuer Certificate :

[Sun Nov 14 02:03:55 2010] Certificate:
    Data:
        Version: 2 (0x1)
        Serial Number: 69 (0x45)
        Signature Algorithm: sha1WithRSAEncryption
        Issuer: C=ES, ST=Barcelona, L=Barcelona, O=isg.upc.es, OU=Cervantes Revocation System, CN=CervantesIssuer
        Validity
            Not Before: Apr 22 06:15:00 2002 GMT
            Not After : Jun  8 21:14:00 2002 GMT
        Subject: C=ES, ST=Barcelona, L=Barcelona, O=isg.upc.es, OU=Cervantes Revocation System, CN=CervantesIssuer
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
            RSA Public Key: (1023 bit)
                Modulus (1023 bit):
                    67:a5:9a:00:8b:a8:41:8b:df:21:b7:9e:3b:8d:d9:
                    e1:ad:3e:75:4f:a1:4f:df:af:ff:f6:56:4a:9e:e4:
                    1a:79:c4:f3:81:a1:ba:54:41:26:ca:93:c2:79:e9:
                    6d:de:5a:f5:56:ff:3a:1f:d3:6d:85:8b:7c:3b:f3:
                    72:e7:a7:d4:51:fe:2d:16:2a:d2:d4:0b:c7:59:b0:
                    63:97:ca:4b:8e:e6:be:98:a7:93:44:0e:27:de:03:
                    c0:dd:eb:5d:e1:90:3c:76:c7:c5:df:2f:0a:37:3d:
                    9a:5b:f5:9a:e8:47:da:56:ef:d4:5c:87:56:1c:ee:
                    11:13:03:3e:01:ac:a1:0b
                Exponent: 65537 (0x10001)
    Signature Algorithm: sha1WithRSAEncryption
        40:04:22:6f:46:6a:10:1b:53:4f:38:20:f5:f9:26:16:f4:1d:
        b8:ea:87:fa:9b:37:68:55:f2:4a:ce:cc:11:26:99:15:0a:bf:
        0a:8c:ce:d6:d5:89:d8:16:9d:cc:c6:64:8f:d1:0f:cb:04:95:
        3f:27:61:7f:76:4c:d0:0f:31:79:29:2b:78:95:12:4f:25:9c:
        a2:71:d1:60:cc:05:06:a0:04:06:13:43:76:a2:d9:76:f4:a5:
        ec:d5:cb:c5:11:94:11:be:93:ef:78:86:4f:87:fd:74:7a:1c:
        ec:23:f1:81:77:72:55:38:fc:51:45:15:dc:92:88:1f:c2:59:
        30:af


[Sun Nov 14 02:03:55 2010] Revocation Time :

[Sun Nov 14 02:03:55 2010] 2010-11-14 02:03:55z

[Sun Nov 14 02:03:55 2010] Revocation Reason :

[Sun Nov 14 02:03:55 2010] 7
```

Trazas de salida al revocar un certificado en modo STANDARD

En el modo TEST (mode=1) las solicitudes de revocación, las expiraciones de certificados y la generación de nuevos clientes son automáticas (aleatorias, dependiendo de los parámetros configurados en el campo [TEST] del fichero de configuración del Servidor).



Trazas de salida de eventos en el modo TEST.

Ejecutando los clientes

Por el lado de los clientes, disponemos de las clases necesarias para poder crear un cliente y poder enviar al servidor solicitudes de revocación y de descarga de CRL. Ambos tipos de clientes (Clientes SCRP y Clientes CRL) disponen de una interfaz gráfica (GUI) y de una ejecución por línea de comandos.

Cliente SCRP

Nos permite enviar una solicitud al servidor para pedir la revocación de un certificado en particular. Para ejecutar este cliente usaremos el script ***/src/client/revokeCliente.py***
Este script tiene dos tipos de ejecución:

- Si lo ejecutamos sin parámetros, abrirá la interfaz gráfica de usuario para configurar la solicitud SCRP.

```
# python revokeClient.py
```

- Sin embargo, también podemos ejecutar el script añadiéndole tres parámetros obligatorios para poder configurar las peticiones de revocación sin pasar por la interfaz gráfica de usuario. Lo ejecutaremos de la siguiente manera:

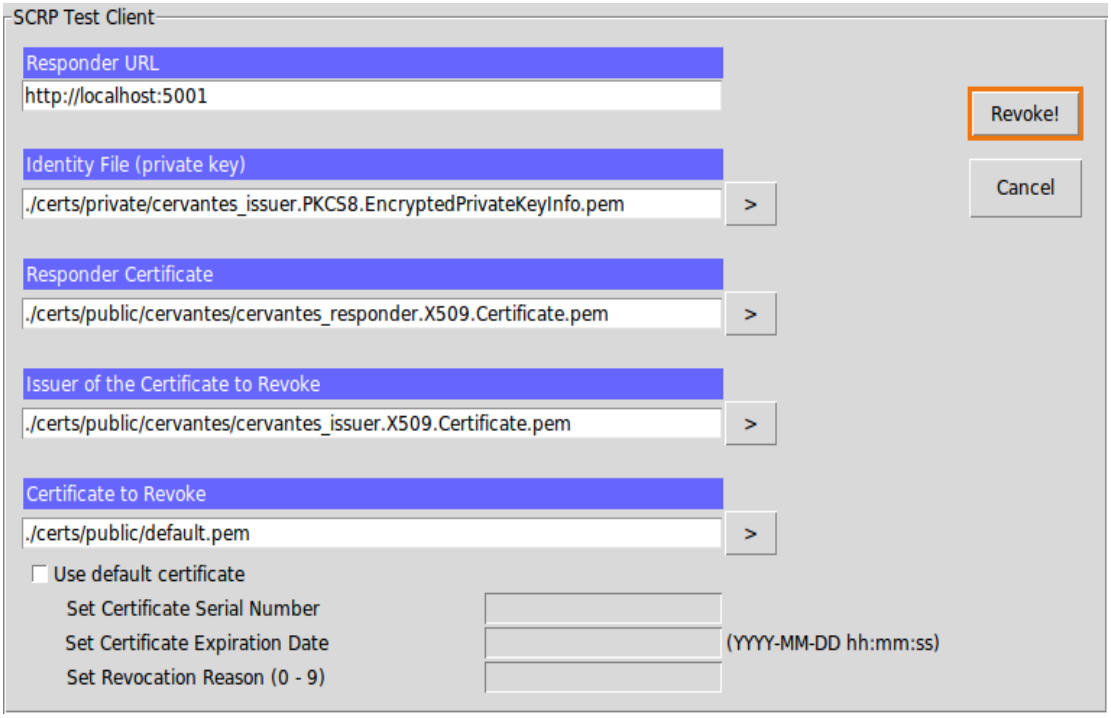
```
# python revokecliente.py <serial_number> <expiration_date>
<revocation_reason>
```

- *serial_number*: número de serie del certificado. Tener en cuenta que al hacer la petición SCRP de esta manera, se utiliza un certificado por defecto (en el cual, se indica que el certificado es emitido por la CA “Cervantes CA”) al cual se le sobreescribe con este número de serie y con la fecha de expiración indicada.
- *expiration_date*: fecha de expiración del certificado. Debe tener el formato: yyyy-mm-dd HH:ii:ss (p.e. 2010-11-14 00:24:15)
- *revocation_reason*: número entero [0,9] que indica la razón de la revocación.

Si alguno de estos parámetros es omitido, o no tiene el formato válido, se indicará el error y se ejecutara la interfaz GUI.

Interfaz Gráfica de Usuario SCRP

La interfaz GUI del cliente SCRP es la siguiente:

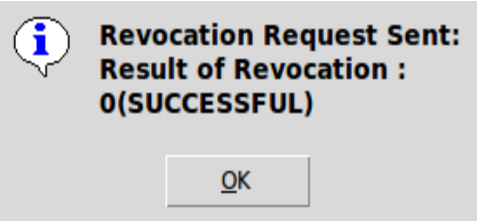


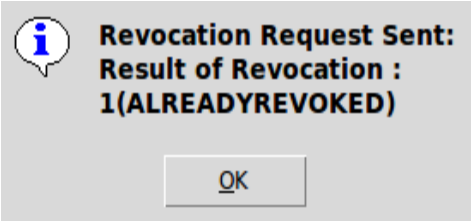
GUI del Cliente SCRP

Sus parámetros configurables son:

- *Responder URL:* indica la IP y el puerto al cual enviar la solicitud de revocación.
- *Identity File:* Clave privada con la que se firmará la solicitud.
- *Responder Certificate:* Ruta del certificado de quien responde la solicitud.
- *Issuer Certificate:* Ruta del certificado del emisor de certificado a revocar.
- *Certificate to Revoke:* Ruta del certificado que se quiere revocar.
- *Si elegimos la opción de certificado por defecto:*
    - Serial Number: número de serie del certificado.
    - Expiration Date: fecha de expiración.
    - Revocation Reason: entero [0, 9] que indica la razón de la revocación.

La respuesta de la revocación es mostrada en un popup:





Cliente CRL

El cliente CRL permite generar una solicitud de descarga de CRL. Para ejecutar este cliente en el modo de línea de comandos ejecutaremos el siguiente script: ***/src/client/runCRLClient.py***
Para ejecutar este cliente en el modo GUI ejecutaremos el siguiente script: ***/src/client/guiCRLClient.py***

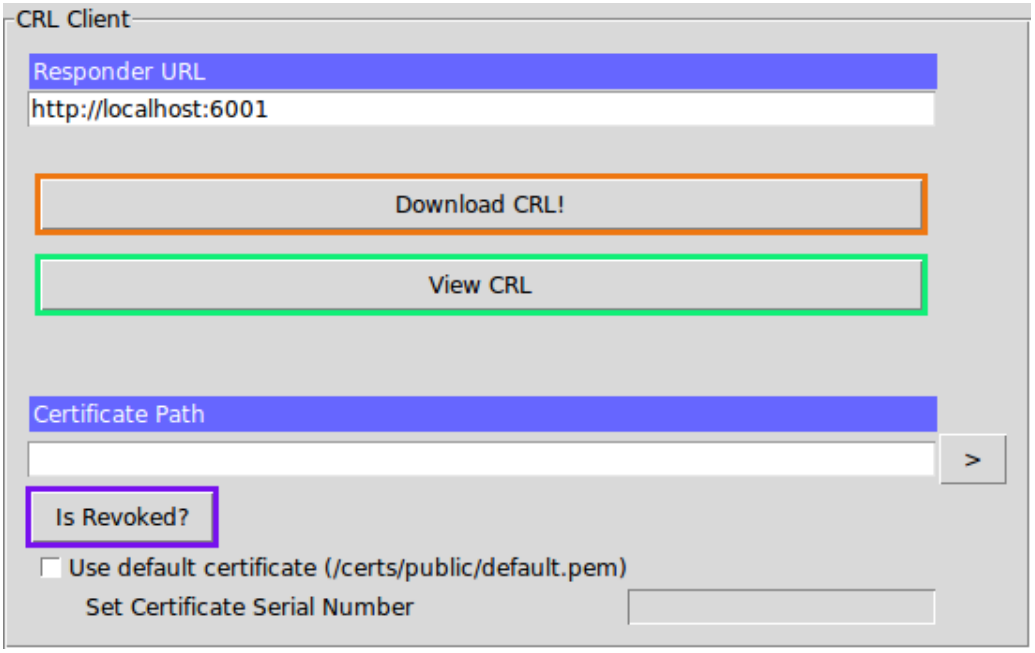
En el modo de ejecución por línea de comandos se crea un hilo que solicitará una CRL y que cada vez que la CRL deje de ser válida intentará descargarla nuevamente del servidor. Es decir, simula un cliente real que está haciendo periódicamente solicitudes de CRL.

```
# python runCRLClient.py
```

```
ronaldo@ronaldo-PC:~/Escritorio/PFC_v0.5.1/src/client$ python runCRLClient.py
SERIAL NUMBER OF CLIENT : 9999
CERTIFICATE OF CLIENT:

Certificate:
    Data:
        Version: 1 (0x0)
        Serial Number: 9999 (0x270f)
        Signature Algorithm: sha1WithRSAEncryption
        Issuer: C=ES, ST=Barcelona, L=Barcelona, O=Retevision, OU=Provision, CN=Isidoro Bernabe
        Validity
            Not Before: Apr 22 06:15:00 2002 GMT
            Not After : Jun  8 21:14:00 2002 GMT
        Subject: C=ES, ST=Barcelona, L=Barcelona, O=Retevision, OU=Provision, CN=Isidoro Bernabe
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
            RSA Public Key: (1024 bit)
                Modulus (1024 bit):
                    00:b9:09:bf:a4:29:07:89:94:55:b6:42:8e:4f:4b:
                    76:f4:f0:0b:4f:07:d8:91:83:3f:0b:eb:fb:08:97:
                    cd:c4:d3:48:ca:c6:82:3c:3c:e8:6c:fa:44:2f:54:
                    c6:4d:2f:26:fb:90:60:dc:9b:c5:9a:10:f8:ec:32:
                    fd:bf:47:2b:0b:f6:7e:d1:a6:4b:41:58:d1:c9:d8:
                    e2:5f:5a:21:63:6b:14:28:ea:81:bb:76:e2:a3:d5:
                    e2:b0:f8:67:33:14:0b:ea:78:5b:30:1b:bf:14:83:
                    7b:37:fb:3d:c1:63:75:32:08:e2:bc:9c:34:cd:80:
                    e3:4e:25:0a:50:7a:f5:56:e7
                Exponent: 65537 (0x10001)
    Signature Algorithm: sha1WithRSAEncryption
        1c:6b:e0:8c:dd:3e:7f:82:7a:0e:fb:3c:6c:52:a0:85:93:9f:
        fd:e1:c1:d3:e5:4f:c5:6e:9c:a3:31:54:8a:34:9e:59:de:a1:
        c2:e4:7f:72:a3:df:23:5a:d4:a3:45:55:43:dd:67:16:da:f2:
        37:b3:0c:f8:d7:15:0c:a6:b0:83:7a:95:5f:5f:6b:56:93:69:
        7f:4c:74:02:3f:24:bb:a2:c8:7b:d5:52:da:26:e4:64:cd:4c:
        b4:17:00:15:1a:76:ef:1a:c9:17:87:a0:52:4f:ae:65:96:28:
        cf:86:b7:9a:64:2e:da:9a:b1:62:11:7c:20:17:74:bf:92:76:
        d5:66

Client connected :: Requesting CRL
REQUEST TO SEND  :  612 bytes
DATOS RECIBIDOS (STATS) :
1000,2
RESPONSE RECEIVED : 310Bytes
Current CRL has 2 revoked certificates.
```

Trazas de cliente CRL ejecutado desde línea de comandos.

En modo GUI, nos mostrará la siguiente interfaz:

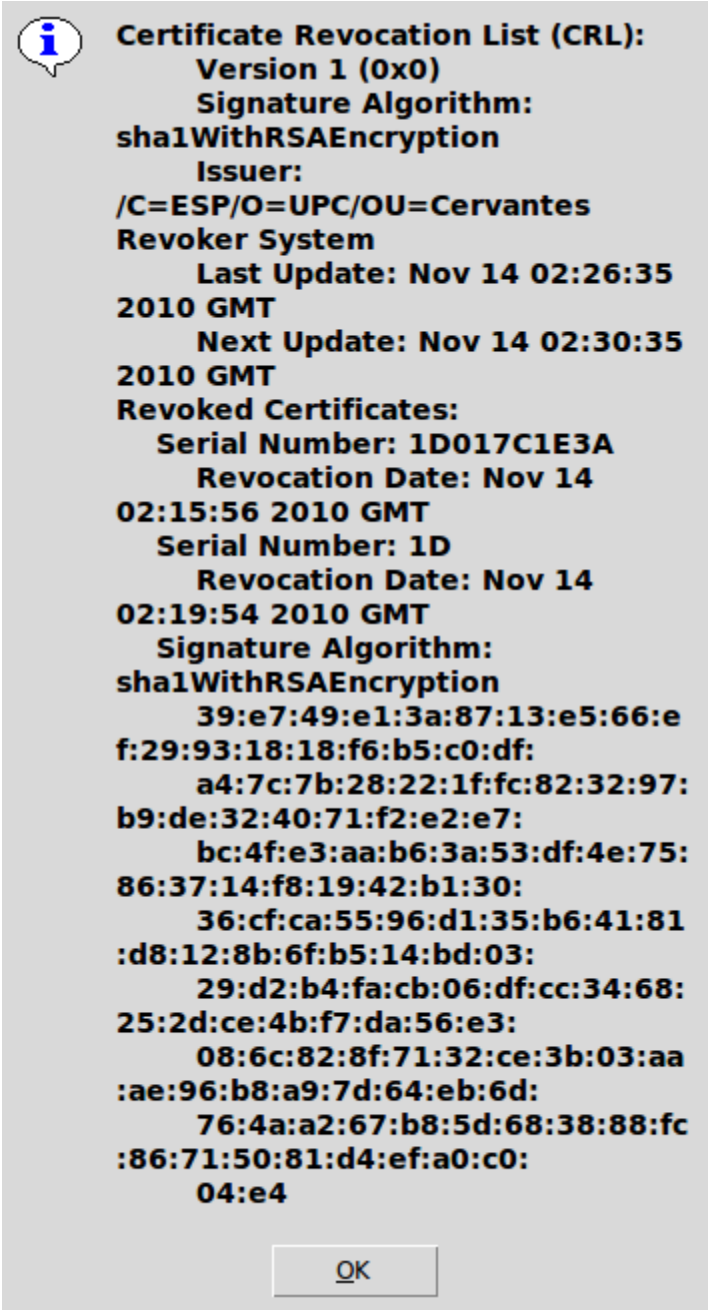


Interfaz GUI de Cliente CRL

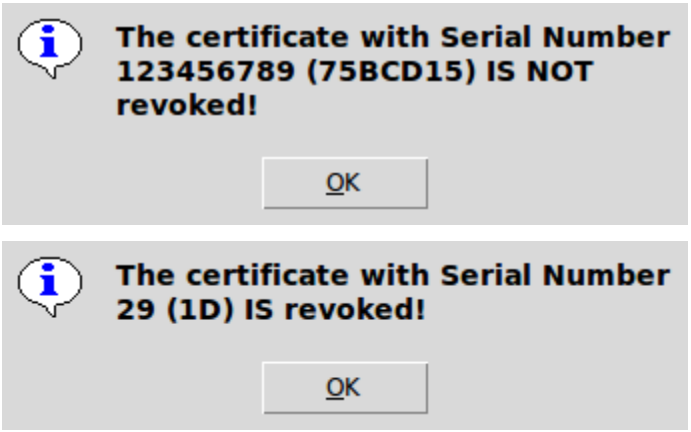
Los parámetros configurables son:

- *Responder URL:* Para indicar la ip y el puerto al cual conectarse para enviar la solicitud de descarga de CRL.
- *Certificate Path:* Ruta de certificado el cual queremos consultar si está revocado o no.
- *Serial Number:* Si elegimos la opción de sobreescribir certificado por defecto, este será el número de serie de dicho certificado.

Podemos visualizar la CRL descargada pulsando al botón **View CRL**:



Podemos consultar si un certificado esta revocado o no pulsando el botón **isRevoked?**:



Apéndice C. Código Fuente del Servidor de la Plataforma de Revocaciones implementada

Clase RevocationsServersManager: /src/server/CentralManagementModule.py

```
import time, sys
import stats.Stats as statistics
import stats.Api as statsAPI
import Properties as prop
import db.DatabaseModule as DB
import scrp.SCRPPortThread as SCRP
import crl.CRLPortThread as CRL
import test.EndSimulation as endSim
import test.TestClient as test_client


class RevocationServersManager():
    #Execution modes
    STANDARD_MODE = 0
    TEST_MODE     = 1
    #Configuration file Path
    SERVER_CONFIG_FILE = './etc/server_conf.txt'
    #Stats Array
    _stats = []
    #Number of clients
    _numberOfCertificates = 0
    _globalCounter = 0
    _testClients = []

    def __init__(self):
      #Setting the simulation start time t0
      self._startTime = time.time()
      #Stats :: Number of Certificates in the system
      self._stats.append(statistics.PopulationSize(self))
      self._stats[0].start()
      #Stats :: Number of Revoked Non-expired Certs
      self._stats.append(statistics.RevokedCertsNumber(self))
      #Get properties from config file
      self.p = prop.Properties(self.SERVER_CONFIG_FILE)
      self.log("Properties loaded from : " + self.SERVER_CONFIG_FILE)
      #Timer to stop the simulation
      self._exitTimer = None
      #Array of controllers (portListener)
      self.portListenerThreads = []
      #Database
      self.dbModule = None
```

```
    def initialize(self):
        """Initialize the Revocations Server an creates the controllers (portListeners) for each SCH
configured in the application."""
        #Simulation mode (0:Standard, 1:Test)
        startMode = int(self.p.general.getProperty('mode'))
        self.log("Start mode : " + str(startMode))
        #Initializing Database
        self.log("Initialize DB ...")
        self.initDB(self.p.database, startMode)
        #Creates one controller thread (portListener) for each SCH in the application (SCRP and CRL)
        self.log("Creating threads for port Listeners ...")
        self.createPortListenerThreads(startMode)

        #In TEST mode, it initializes the simulation parameters read from config file
        if startMode == self.TEST_MODE:
            self.initializeTestMode()
        elif startMode == self.STANDARD_MODE:
            self._numberOfCertificates = int(self.p.general.getProperty('numberOfCertificates'))
            self._stats[0].statsLog(self._numberOfCertificates)
        #Initialize the API listener of statistics
        self.statisticsAPI = statsAPI.Api(self)
        self.statisticsAPI.start()
        #Initialize all threads of all controllers registered as portListeners
        self.startPortListenerThreads()


    def initializeTestMode(self):
        """Executes all tasks needed to start the application in TEST_MODE."""
        #Create thread to End current simulation according to "testDuration" config parameter
        self._exitTimer = endSim.EndSimulation(self, int(self.p.test.getProperty('testDuration')))
        self._exitTimer.start()
        #Parameters of [Stage]
        stage = None
        #Leer del archivo de configuracion el algoritmo a utilizar para simular conexiones/desconexiones
        disconnectionsAlg = int(self.p.test.getProperty("disconnectionsAlgorithm"))
        if disconnectionsAlg == 0 or disconnectionsAlg == 1:
            stage = self.p.stage
        #Create the number of certificates specified in config file
        numberOfCerts = int(self.p.general.getProperty('numberOfCertificates'))
        for i in range(0, numberOfCerts):
            self.generateClient()


    def createPortListenerThreads(self, startMode):
        """Crea un hilo por cada servidor en el archivo de configuracion, esuchando las peticiones de los
clientes en un puerto especifico."""
        #Read test parameters (only in TEST_MODE)
        testProps = None
        if startMode == self.TEST_MODE:
            testProps = self.p.test
       #Append all servers configured in the config file
        availableServers = []
        if self.p.scrp:
            availableServers.append(self.p.scrp)
        if self.p.crl:
            availableServers.append(self.p.crl)
```

```
        #Creates an array to control the portListeners of each server
        for currServer in availableServers:
            typeOfServer = currServer.getProperty("type")
            serverName   = currServer.getProperty("name")
            if typeOfServer == "SCRP":
                self.log("Creating server " + typeOfServer + " --> " + serverName)
                obj = SCRP.SCRPPortThread(self, currServer, self.dbModule, startMode, testProps)
                self.portListenerThreads.append(obj)
            elif typeOfServer == "CRL":
                self.log("Creating server " + typeOfServer + " --> " + serverName)
                obj = CRL.CRLPortThread(self, currServer, self.dbModule, startMode)
                self.portListenerThreads.append(obj)
            else:
                #Type of server does not exists
                self.log("The type of server *" + str(typeOfServer) + "* was not found.")


    def generateClient(self):
        """Creates a new client object, adds it to the array of clients, and starts its thread (for mobility
purposes)."""
        self.log("Creating client id# " + str(self._globalCounter))
        cl = test_client.TestClient(self._globalCounter)
        self._testClients.append(cl)
        self._numberOfCertificates += 1
        self._globalCounter += 1
        self._stats[0].statsLog(self._numberOfCertificates)


    def startPortListenerThreads(self):
        """Starts the thread of all the port listeners Servers listed in config file"""
        i = 0 #Cont aux variable
        for portListenerThread in self.portListenerThreads:
            self.log("Starting Server " + str(i) + " ... ")
            portListenerThread.start()
            i+=1


    def initDB(self, dbProperties, startMode):
        """Creates database connector object and initialize the DB"""
        #Initialize the DB connection module with the parameters set in the config file
        self.dbModule = DB.DatabaseModule()
        self.dbModule.initialize(dbProperties)
        #Deletes all records in database
        if startMode == self.TEST_MODE:
            #In test mode, all records pre-existent in the DB are deleted
            self.dbModule.newDBSession().deleteAllRecords()
            self.log("Test Mode: Database intialized with 0 records.")
        elif startMode == self.STANDARD_MODE:
            #In standard mode, the DB is cleaned depending on the config file “clearDatabaseContents"
            if (dbProperties.getProperty('clearDatabaseContents') == "TRUE"):
                    self.dbModule.newDBSession().deleteAllRecords()
                    self.log("Standard mode : Database cleaned! (clearDatabaseContents = TRUE)")


    def log(self, msg=''):
        """Writes logs in the server_manager log file.""
```

```
        f = open('./log/server_manager.log', 'a')
        f.write("[" + time.asctime() + "] " + msg + '\n')
        f.close()
        print (msg);


    def exit_program(self):
        """Terminates the execution of the application"""
        for i in range(len(self.portListenerThreads)):
            self.portListenerThreads[i].stop()
            self.portListenerThreads[i].join()
        self._stats[0].stop()
        self._stats[0].join()

        self.statisticsAPI.stop()
        self.statisticsAPI.join()

        self.log("Exit application!!! BYE!")
        sys.exit(0)
```

Clase Properties: /src/server/Properties.py

```
class Properties:
    def __init__(self, path):
        self.general  = PropertyList()
        self.database = PropertyList()
        self.stage    = PropertyList()
        self.test     = PropertyList()
        self.scrp     = PropertyList()
        self.crl      = PropertyList()
        #Open configuration file and read contents
        f = open(path, 'r')
        #Trim lines with no data
        self.file = self.trimFile(f)
        #Load config parameters in properties arrays
        self.loadConfig()
        #Close file
        f.close()


    def trimFile(self, f):
        lines = []
        for x in f.readlines():
            s = x.strip()
            if s != "":
                lines.append(s)
        return lines


    def loadConfig(self):
        propType = ""
```

```
            for y in self.file:
                if y.startswith("["):
                    propType = y[1:len(y)-1]
                else:
                    property = Property(y)
                    self.addProperty(property, propType)


        def addProperty(self, prop, type):
            if type == "General":
                self.general.add(prop)
            elif type == "Database":
                self.database.add(prop)
            elif type == "Stage":
                self.stage.add(prop)
            elif type == "Test":
                self.test.add(prop)
            elif type == "SCRP":
                self.scrp.add(prop)
            elif type == "CRL":
                self.crl.add(prop)


class PropertyList:
        def __init__(self):
            self.lista = []

        def add(self, prop):
            self.lista.append(prop)

        def getProperty(self, atribute, default=None):
            for x in self.lista:
                if x.getParameter() == atribute:
                    return x.getValue()
            return default


class Property:
        def __init__(self, line):
            b = line.split("=")
            self.parameter = b[0].strip()
            self.value     = b[1].strip()

        def getParameter(self):
            return self.parameter

        def getValue(self):
            return self.value
```

Clase X509Certificate: /src/server/asn1/x509/X509Certificate.py

```
from pyasn1.type import univ, char, useful

class X509Certificate:
        asn1Spec = None
        ASN1Cert = None

        def __init__(self):
                self.asn1Spec = self.getX509CertificateInstance()
                self.ASN1Cert = self.getX509CertificateInstance()

        def getX509CertificateInstance(self):
                tbs = univ.namedtype.NamedType('tbsCertificate', TBSCertificate().getAsn1Spec())
                alg = univ.namedtype.NamedType('signatureAlgorithm',
AlgorithmIdentifier().getAsn1Spec())
                sig = univ.namedtype.NamedType('signature', univ.BitString())
                components = univ.namedtype.NamedTypes(tbs, alg, sig)
                asn1 = univ.Sequence(componentType=components)
                return asn1

        def getAsn1Spec(self):
                return self.asn1Spec

        def getASN1Cert(self):
                return self.ASN1Cert

        def setTbsCertificate(self, x):
                self.ASN1Cert.setComponentByName('tbsCertificate', x)

        def getTbsCertificate(self):
                return self.ASN1Cert.getComponentByName('tbsCertificate')

        def setSignatureAlgorithm(self, x):
                self.ASN1Cert.setComponentByName('signatureAlgorithm', x)

        def getSignatureAlgorithm(self):
                return self.ASN1Cert.getComponentByName('signatureAlgorithm')

        def setSignature(self, x):
                self.ASN1Cert.setComponentByName('signature', x)

        def getSignature(self):
                return self.ASN1Cert.getComponentByName('signature')

        def createX509Certificate(self, tbs, salg, sig):
                self.ASN1Cert = self.getAsn1Spec()
                self.setTbsCertificate(tbs)
                self.setSignatureAlgorithm(salg)
                self.setSignature(sig)
                return self.getASN1Cert()


class TBSCertificate:
```

```
        asn1Spec = None
        ASN1TBSCert = None

        def __init__(self):
                self.asn1Spec    = self.getTBSCertficateInstance()
                self.ASN1TBSCert = self.getTBSCertficateInstance()

        def getTBSCertficateInstance(self):
                ver = univ.namedtype.NamedType('version', univ.Integer())
                snu = univ.namedtype.NamedType('serialNumber', univ.Integer())
                sig = univ.namedtype.NamedType('signature', AlgorithmIdentifier().getAsn1Spec())
                iss = univ.namedtype.NamedType('issuer', Name().getAsn1Spec())
                val = univ.namedtype.NamedType('validity', Validity().getAsn1Spec())
                sub = univ.namedtype.NamedType('subject', Name().getAsn1Spec())
                spk = univ.namedtype.NamedType('subjectPublicKeyInfo',
SubjectPublicKeyInfo().getAsn1Spec())
                components = univ.namedtype.NamedTypes(ver, snu, sig, iss, val, sub, spk)
                asn1 = univ.Sequence(componentType=components)
                return asn1

        def getAsn1Spec(self):
                return self.asn1Spec

        def getASN1TBSCert(self):
                return self.ASN1TBSCert

        def setVersion(self, x):
                self.ASN1TBSCert.setComponentByName('version', x)

        def getVersion(self):
                return self.ASN1TBSCert.getComponentByName('version')

        def setSerialNumber(self, x):
                self.ASN1TBSCert.setComponentByName('serialNumber', x)

        def getSerialNumber(self):
                return self.ASN1TBSCert.getComponentByName('serialNumber')

        def setSignature(self, x):
                self.ASN1TBSCert.setComponentByName('signature', x)

        def getSignature(self):
                return self.ASN1TBSCert.getComponentByName('signature')

        def setIssuer(self, x):
                self.ASN1TBSCert.setComponentByName('issuer', x)

        def getIssuer(self):
                return self.ASN1TBSCert.getComponentByName('issuer')

        def setValidity(self, x):
                self.ASN1TBSCert.setComponentByName('validity', x)

        def getValidity(self):
                return self.ASN1TBSCert.getComponentByName('validity')
```

```
        def setSubject(self, x):
                self.ASN1TBSCert.setComponentByName('subject', x)

        def getSubject(self, x):
                return self.ASN1TBSCert.getComponentByName('subject')

        def setSubjectPublicKeyInfo(self, x):
                self.ASN1TBSCert.setComponentByName('subjectPublicKeyInfo', x)

        def getSubjectPublicKeyInfo(self):
                return self.ASN1TBSCert.getComponentByName('subjectPublicKeyInfo')

        def createTBSCertificate(self, v, sn, sig, iss, val, sub, spk):
                self.ASN1TBSCert = self.getAsn1Spec()
                self.setVersion(v)
                self.setSerialNumber(sn)
                self.setSignature(sig)
                self.setIssuer(iss)
                self.setValidity(val)
                self.setSubject(sub)
                self.setSubjectPublicKeyInfo(spk)
                return self.getASN1TBSCert()


class Name:
        asn1Spec = None
        ASN1Name = None

        def __init__(self):
                self.asn1Spec = self.getIssuerInstance()
                self.ASN1Name = self.getIssuerInstance()

        def getIssuerInstance(self):
                asn1 = univ.Sequence()
                asn1.setComponentByPosition(0, univ.Set().setComponentByPosition(0,
univ.Sequence(componentType=univ.namedtype.NamedTypes(univ.namedtype.OptionalNamedType('oi
dC',univ.ObjectIdentifier()), univ.namedtype.NamedType('C', char.PrintableString())))))
                asn1.setComponentByPosition(1, univ.Set().setComponentByPosition(0,
univ.Sequence(componentType=univ.namedtype.NamedTypes(univ.namedtype.OptionalNamedType('oi
dST',univ.ObjectIdentifier()), univ.namedtype.NamedType('ST', char.PrintableString())))))
                asn1.setComponentByPosition(2, univ.Set().setComponentByPosition(0,
univ.Sequence(componentType=univ.namedtype.NamedTypes(univ.namedtype.OptionalNamedType('oi
dL',univ.ObjectIdentifier()), univ.namedtype.NamedType('L', char.PrintableString())))))
                asn1.setComponentByPosition(3, univ.Set().setComponentByPosition(0,
univ.Sequence(componentType=univ.namedtype.NamedTypes(univ.namedtype.OptionalNamedType('oi
dO',univ.ObjectIdentifier()), univ.namedtype.NamedType('O', char.PrintableString())))))
                asn1.setComponentByPosition(4, univ.Set().setComponentByPosition(0,
univ.Sequence(componentType=univ.namedtype.NamedTypes(univ.namedtype.OptionalNamedType('oi
dOU',univ.ObjectIdentifier()), univ.namedtype.NamedType('OU', char.PrintableString())))))
                asn1.setComponentByPosition(5, univ.Set().setComponentByPosition(0,
univ.Sequence(componentType=univ.namedtype.NamedTypes(univ.namedtype.OptionalNamedType('oi
dCN',univ.ObjectIdentifier()), univ.namedtype.NamedType('CN', char.PrintableString())))))
                return asn1

        def setASN1(self, asn):
        self.setC(asn.getComponentByPosition(0).getComponentByPosition(0).getComponentByPositio
n(1))
```

```
        self.setST(asn.getComponentByPosition(1).getComponentByPosition(0).getComponentByPositi
on(1))

        self.setL(asn.getComponentByPosition(2).getComponentByPosition(0).getComponentByPositio
n(1))

        self.setO(asn.getComponentByPosition(3).getComponentByPosition(0).getComponentByPositio
n(1))

        self.setOU(asn.getComponentByPosition(4).getComponentByPosition(0).getComponentByPositi
on(1))

        self.setCN(asn.getComponentByPosition(5).getComponentByPosition(0).getComponentByPositi
on(1))


        def getAsn1Spec(self):
                return self.asn1Spec

        def getASN1Name(self):
                return self.ASN1Name

        def setC(self, x):

        self.ASN1Name.getComponentByPosition(0).getComponentByPosition(0).setComponentByNam
e('C', x)

        def getC(self):
                return
self.ASN1Name.getComponentByPosition(0).getComponentByPosition(0).getComponentByName('C')

        def setST(self, x):

        self.ASN1Name.getComponentByPosition(1).getComponentByPosition(0).setComponentByNam
e('ST', x)

        def getST(self):
                return
self.ASN1Name.getComponentByPosition(1).getComponentByPosition(0).getComponentByName('ST')

        def setL(self, x):

        self.ASN1Name.getComponentByPosition(2).getComponentByPosition(0).setComponentByNam
e('L', x)

        def getL(self):
                return
self.ASN1Name.getComponentByPosition(2).getComponentByPosition(0).getComponentByName('L')

        def setO(self, x):

        self.ASN1Name.getComponentByPosition(3).getComponentByPosition(0).setComponentByNam
e('O', x)

        def getO(self):
```

```
                return
self.ASN1Name.getComponentByPosition(3).getComponentByPosition(0).getComponentByName('O')

        def setOU(self, x):

        self.ASN1Name.getComponentByPosition(4).getComponentByPosition(0).setComponentByNam
e('OU', x)

        def getOU(self):
                return
self.ASN1Name.getComponentByPosition(4).getComponentByPosition(0).getComponentByName('OU')

        def setCN(self, x):

        self.ASN1Name.getComponentByPosition(5).getComponentByPosition(0).setComponentByNam
e('CN', x)

        def getCN(self):
                return
self.ASN1Name.getComponentByPosition(5).getComponentByPosition(0).getComponentByName('CN')

        def getName(self):
                ret = "C=" + self.getC() + " ST=" + self.getST() + " L=" + self.getL() + " O=" + self.getO() +
" OU=" + self.getOU() + " CN=" + self.getCN()
                return ret

        def createIssuer(self, c, st, l, o, ou, cn):
                self.ASN1Name = self.getAsn1Spec()
                self.setC(c)
                self.setST(st)
                self.setL(l)
                self.setO(o)
                self.setOU(ou)
                self.setCN(cn)
                return self.getASN1Name()


class Validity:
        #ASN.1 Spec
        asn1Spec    = None
        ASN1Validity = None

        def __init__(self):
                self.asn1Spec = self.getValidityInstance()
                self.ASN1Validity = self.getValidityInstance()

        def getValidityInstance(self):
                nb = univ.namedtype.NamedType('notBefore', useful.UTCTime())
                na = univ.namedtype.NamedType('notAfter', useful.UTCTime())
                components = univ.namedtype.NamedTypes(nb, na)
                asn1 = univ.Sequence(componentType=components)
                return asn1

        def getAsn1Spec(self):
                return self.asn1Spec

        def getASN1Validity(self):
```

```
                return self.ASN1Validity

        def setNotBefore(self, x):
                self.ASN1Validity.setComponentByName('notBefore', x)

        def getNotBefore(self):
                return self.ASN1Validity.getComponentByName('notBefore')

        def setNotAfter(self, x):
                self.ASN1Validity.setComponentByName('notAfter', x)

        def getNotAfter(self, x):
                return self.ASN1Validity.getComponentByName('notAfter')

        def createValidity(self, nb, na):
                self.ASN1Validity = self.getAsn1Spec()
                self.setNotBefore(nb)
                self.setNotAfter(na)
                return self.getASN1Validity()


class SubjectPublicKeyInfo:
        asn1Spec     = None
        ASN1SubjectPKInfo = None

        def __init__(self):
                self.asn1Spec        = self.getSubjectPublicKeyInfoInstance();
                self.ASN1SubjectPKInfo = self.getSubjectPublicKeyInfoInstance();

        def getSubjectPublicKeyInfoInstance(self):
                alg = univ.namedtype.NamedType('algorithm', AlgorithmIdentifier().getAsn1Spec())
                spk = univ.namedtype.OptionalNamedType('subjectPublicKey', univ.BitString())
                components = univ.namedtype.NamedTypes(alg, spk)
                asn1 = univ.Sequence(componentType=components)
                return asn1

        def getAsn1Spec(self):
                return self.ASN1SubjectPKInfo

        def getASN1SubjectPKInfo(self):
                return self.ASN1SubjectPKInfo

        def setAlgorithm(self, a):
                self.ASN1SubjectPKInfo.setComponentByName('algorithm', a)

        def getAlgorithm(self):
                return self.ASN1SubjectPKInfo.getComponentByName('algorithm')

        def setSubjectPublicKey(self, spk):
                self.ASN1SubjectPKInfo.setComponentByName('subjectPublicKey',spk)

        def getSubjectPublicKey(self):
                return self.ASN1SubjectPKInfo.getComponentByName('subjectPublicKey')

        def createValidity(self, nb, na):
                self.ASN1SubjectPKInfo = self.getAsn1Spec()
```

```
                self.setAlgorithm(nb)
                self.setSubjectPublicKey(na)
                return self.getASN1SubjectPKInfo()


class AlgorithmIdentifier:
        asn1Spec        = None
        ASN1AlgIdentifier = None

        def __init__(self):
                self.asn1Spec = self.getAlgorithmIdentifierInstance()
                self.ASN1AlgIdentifier = self.getAlgorithmIdentifierInstance()

        def getAlgorithmIdentifierInstance(self):
                a = univ.namedtype.NamedType('algorithm', univ.ObjectIdentifier())
                p = univ.namedtype.OptionalNamedType('parameters', univ.Null())
                components = univ.namedtype.NamedTypes(a, p)
                asn1 = univ.Sequence(componentType=components)
                return asn1

        def getAsn1Spec(self):
                return self.asn1Spec

        def getASN1AlgIdentifier(self):
                return self.ASN1AlgIdentifier

        def setAlgorithm(self, a):
                self.ASN1AlgIdentifier.setComponentByName('algorithm', a)

        def getAlgorithm(self):
                return self.ASN1AlgIdentifier.getComponentByName('algorithm')

        def setParameters(self, p):
                self.ASN1AlgIdentifier.setComponentByName('parameters',p)

        def getParameters(self):
                return self.ASN1AlgIdentifier.getComponentByName('parameters')

        def createValidity(self, a, p):
                self.ASN1AlgIdentifier = self.getAsn1Spec()
                self.setAlgorithm(a)
                self.setParameters(p)
                return self.getASN1AlgIdentifier()
```

Clase CRL: /src/server/asn1/crl/CRL.py

```
from pyasn1.type import univ, char, useful

class CRL:
    def __init__(self):
        self._asn1 = univ.Sequence()
        #CRL Fields
```

```
        self._tbsCertList       = None
        self._signatureAlgorithm = None
        self._signature         = None

    def setTbsCertList(self, ObjTbsCertList):
        self._tbsCertList = ObjTbsCertList.getAsn1()
        self._asn1.setComponentByPosition(0, self._tbsCertList)

    def getTbsCertList(self):
        return self._tbsCertList

    def setSignatureAlgorithm(self, ObjAlgorithmIdentifier):
        self._signatureAlgorithm = ObjAlgorithmIdentifier.getAsn1()
        self._asn1.setComponentByPosition(1, self._signatureAlgorithm)

    def getSignatureAlgorithm(self):
        return self._signatureAlgorithm

    def setSignature(self, ObjSignature):
        self._signature = ObjSignature.getAsn1()
        self._asn1.setComponentByPosition(2, self._signature)

    def getSignature(self):
        return self._signature

    def getAsn1(self):
        return self._asn1


class TBSCertList:
    def __init__(self):
        self._asn1 = univ.Sequence()
        #TBSCertList Fields
        self._algorithmIdentifier = None
        self._issuerName        = None
        self._thisUpdate        = None
        self._nextUpdate        = None
        self._revokedCetificates  = univ.Sequence()

    def setAlgorithmIdentifier(self, ObjAlgorithmIdentifier):
        self._algorithmIdentifier = ObjAlgorithmIdentifier.getAsn1()
        self._asn1.setComponentByPosition(0, self._algorithmIdentifier)

    def getAlgorithmIdentifier(self):
        return self._algorithmIdentifier

    def setIssuer(self, ObjName):
        self._issuerName = ObjName.getAsn1()
        self._asn1.setComponentByPosition(1, self._issuerName)

    def getIssuer(self):
        return self._issuerName

    def setThisUpdate(self, thisUpdate):
        self._thisUpdate = useful.UTCTime(str(thisUpdate))
        self._asn1.setComponentByPosition(2, self._thisUpdate)
```

```
    def getThisUpdate(self):
        return self._thisUpdate

    def setNextUpdate(self, nextUpdate):
        self._nextUpdate = useful.UTCTime(str(nextUpdate))
        self._asn1.setComponentByPosition(3, self._nextUpdate)

    def getNextUpdate(self):
        return self._nextUpdate

    def addRevokedCertificate(self, userCertificate, revocationDate):
        revokedCert = univ.Sequence().setComponentByPosition(0,
univ.Integer(int(userCertificate))).setComponentByPosition(1, useful.UTCTime(str(revocationDate)))
        self._revokedCetificates.setComponentByPosition(self._revokedCetificates.__len__(),
value=revokedCert)

    def setRevokedCertificates(self):
        self._asn1.setComponentByPosition(4, self._revokedCetificates)

    def clearRevokedCertificates(self):
        self._revokedCetificates = univ.Sequence()

    def getRevokedCertificates(self):
        return self._revokedCetificates

    def getAsn1(self):
        return self._asn1


class Name:
    def __init__(self):
        self._asn1 = univ.Sequence()
        self._C    = None
        self._O    = None
        self._OU   = None

    def setC(self, c):
        self._C = univ.Set().setComponentByPosition(0, univ.Sequence().setComponentByPosition(0,
univ.ObjectIdentifier('2.5.4.6')).setComponentByPosition(1, char.PrintableString(str(c))))
        self._asn1.setComponentByPosition(0, self._C)

    def setO(self, o):
        self._O = univ.Set().setComponentByPosition(0, univ.Sequence().setComponentByPosition(0,
univ.ObjectIdentifier('2.5.4.10')).setComponentByPosition(1, char.PrintableString(str(o))))
        self._asn1.setComponentByPosition(1, self._O)

    def setOU(self, ou):
        self._OU = univ.Set().setComponentByPosition(0, univ.Sequence().setComponentByPosition(0,
univ.ObjectIdentifier('2.5.4.11')).setComponentByPosition(1, char.PrintableString(str(ou))))
        self._asn1.setComponentByPosition(2, self._OU)

    def getC(self):
        return self._C

    def getO(self):
        return self._O
```

```
    def getOU(self):
        return self._OU

    def getAsn1(self):
        return self._asn1

    def getName(self):
        return "C=" + self._C.getComponentByPosition(0).getComponentByPosition(1) + " O=" +
self._O.getComponentByPosition(0).getComponentByPosition(1) + " OU=" +
self._OU.getComponentByPosition(0).getComponentByPosition(1)


class Signature:
    def __init__(self):
        self._asn1 = None

    def setSignature(self, signature):
        self._asn1 = univ.BitString("'"+signature+"'B")

    def getAsn1(self):
        return self._asn1


class AlgorithmIdentifier:
    def __init__(self):
        self._asn1      = univ.Sequence()
        self._algorithm  = None
        self._parameters = None

    def setAlgorithm(self, OID):
        self._algorithm = univ.ObjectIdentifier(OID)
        self._asn1      = self._asn1.setComponentByPosition(0, self._algorithm)

    def setParameters(self, parameters):
        self._parameters = univ.Null(parameters)
        self._asn1       = self._asn1.setComponentByPosition(1, self._parameters)

    def getAsn1(self):
        return self._asn1
```

Clase SCRPRequest: /src/server/asn1/scrp/SCRPRequest.py

```
from pyasn1.type import univ

class SCRPRequest:
    def __init__(self):
        asn1 = univ.Sequence()
        self._asn1Spec = asn1

    def setTbsRequest(self, tbsrequest):
        self._asn1Spec = self._asn1Spec.setComponentByPosition(0,tbsrequest)
```

```
    def setSignature(self, signature):
        self._asn1Spec = self._asn1Spec.setComponentByPosition(1,signature)

    def getAsn1Spec(self):
        return self._asn1Spec


class Request:

    def __init__(self):
        asn1 = univ.Sequence()
        self._asn1Spec = asn1

    def setCertToRev(self, certToRev):
        self._asn1Spec = self._asn1Spec.setComponentByPosition(0,certToRev)

    def setIssuerCert(self, issuerCert):
        self._asn1Spec = self._asn1Spec.setComponentByPosition(1,issuerCert)

    def setRevocationTime(self, revTime):
        self._asn1Spec = self._asn1Spec.setComponentByPosition(2,revTime)

    def setRevocationReason(self, revReason):
        r0 = ("unspecified",        0)
        r1 = ("keyCompromise",      1)
        r2 = ("cACompromise",       2)
        r3 = ("affiliationChanged", 3)
        r4 = ("superseded",         4)
        r5 = ("cessationOfOperation", 5)
        r6 = ("certificateHold",    6)
        r7 = ("removeFromCRL",      7)
        r8 = ("privilegeWithdrawn", 8)
        r9 = ("aACompromise",       9)
        vals  = univ.namedval.NamedValues(r0,r1,r2,r3,r4,r5,r6,r7,r8,r9)
        self._asn1Spec = self._asn1Spec.setComponentByPosition(3,univ.Enumerated(value=revReason,
tagSet=None, subtypeSpec=None, namedValues=vals))

    def getAsn1Spec(self):
        return self._asn1Spec


class AlgorithmIdentifier:
    def __init__(self):
        asn1 = univ.Sequence()
        self._asn1Spec = asn1

    def setAlgorithm(self, OID):
        self._asn1Spec = self._asn1Spec.setComponentByPosition(0,univ.ObjectIdentifier(OID))

    def setParameters(self, params):
        self._asn1Spec = self._asn1Spec.setComponentByPosition(1,params)

    def getAsn1Spec(self):
        return self._asn1Spec
```

```
class Signature:
    def __init__(self):
        asn1 = univ.Sequence()
        self._asn1Spec = asn1

    def setSignatureAlgorithm(self, algID):
        self._asn1Spec = self._asn1Spec.setComponentByPosition(0,algID)

    def setSignature(self, signature):
        self._asn1Spec = self._asn1Spec.setComponentByPosition(1,univ.BitString("'"+signature+"'B"))

    def getAsn1Spec(self):
        return self._asn1Spec
```

Clase SCRPResponse: /src/server/asn1/scrp/SCRPResponse.py

```
from pyasn1.type import univ
import asn1.x509.X509Certificate as cert

class SCRPResponse:
    def __init__(self):
        asn1 = univ.Sequence()
        self._asn1Spec = asn1

    def setTbsResponse(self, tbsresponse):
        self._asn1Spec = self._asn1Spec.setComponentByPosition(0, tbsresponse)

    def setSignature(self, signature):
        self._asn1Spec = self._asn1Spec.setComponentByPosition(1, signature)

    def getAsn1Spec(self):
        return self._asn1Spec


class Response:
    def __init__(self):
        asn1 = univ.Sequence()
        self._asn1Spec = asn1

    def setCertToRev(self, certToRev):
        self._asn1Spec = self._asn1Spec.setComponentByPosition(0, certToRev)

    def setResult(self, result):
        self._asn1Spec = self._asn1Spec.setComponentByPosition(1, univ.Enumerated(result))

    def getAsn1Spec(self):
        return self._asn1Spec


class AlgorithmIdentifier:
    def __init__(self):
```

```
        asn1 = univ.Sequence()
        self._asn1Spec = asn1

    def setAlgorithm(self, OID):
        self._asn1Spec = self._asn1Spec.setComponentByPosition(0,univ.ObjectIdentifier(OID))

    def setParameters(self, params):
        self._asn1Spec = self._asn1Spec.setComponentByPosition(1,params)

    def getAsn1Spec(self):
        return self._asn1Spec


class Signature:
    def __init__(self):
        asn1 = univ.Sequence()
        self._asn1Spec = asn1

    def setSignatureAlgorithm(self, algID):
        self._asn1Spec = self._asn1Spec.setComponentByPosition(0,algID)

    def setSignature(self, signature):
        self._asn1Spec = self._asn1Spec.setComponentByPosition(1,univ.BitString("'"+signature+"'B"))

    def getAsn1Spec(self):
        return self._asn1Spec
```

Clase CacheUpdater: /src/server/crl/CacheUpdater.py

```
#
# Class CacheUpdater
# extends from threading.Thread
#
# Updates the CRL Cache with all the revoked Certificates contained from DB.
#
import threading, time, binascii, base64
import asn1.crl.CRL as _crl
from M2Crypto import EVP
from pyasn1.codec.der import encoder

class CacheUpdater(threading.Thread):
    OID_RSAwithSHA1 = '1.2.840.113549.1.1.5'

    def __init__(self, portThread, waitTime):
        threading.Thread.__init__(self)
        self._waitTime = waitTime
        self._portThread = portThread
        self._thisUpdate = 0
        self._nextUpdate = 0
        self._addToCRL   = None

    def run(self):
```

```
        while self._portThread._listening:
            t = time.time()
            self._thisUpdate = self.getUTCTime(currentTime=t, add=0)
            self._nextUpdate = self.getUTCTime(currentTime=t, add=self._portThread._validityPeriod)
            #Get all revoked certificates from the Database
            revokedCerts = self._portThread._database.newDBSession().listAllRecords()
            #Create an array containing the data to insert in the CRL for each revoked certificate
[serial_numer, revocationDate]
            self._addToCRL = []
            for revCert in revokedCerts:
                self._addToCRL.append([int(revCert.getSerial_number()),
str(revCert.getRevocation_time())])
            #Generate the new CRL and updated the CRL of the CRLPortThread controller
            __reg1 = self.generateCRL()
            self._portThread._cacheCRL = __reg1
            self._portThread.log("Cache Updater Thread --> CRL updated")
            #Wait until the next update of the CRL
            time.sleep(self._waitTime)


    def getUTCTime(self, currentTime = None, add=0):
        """Returns actual date in UTC format. If receives one parameter of type Number, it adds it to
current date (in seconds) before returning it."""
        if currentTime is None:
            currentTime = time.time()
        t = currentTime
        t += add
        return time.strftime('%y%m%d%H%M%SZ', time.localtime(t))

    def generateCRL(self):
        crl    = _crl.CRL()
        #Create algorithm identifier Object
        sigAlg = _crl.AlgorithmIdentifier()
        sigAlg.setAlgorithm(self.OID_RSAwithSHA1)
        sigAlg.setParameters('')
        #Add Field "SignatureAlgorithm" to the CRL
        crl.setSignatureAlgorithm(sigAlg)
        #Create TBSCertList object
        ###-> Set Algorithm Identifier Field
        tbs = _crl.TBSCertList()
        tbs.setAlgorithmIdentifier(sigAlg)
        ###-> Set Issuer Name Field
        issuer = _crl.Name()
        issuer.setC('ESP')
        issuer.setO('UPC')
        issuer.setOU('Cervantes Revoker System')
        tbs.setIssuer(issuer)
        ###-> Set thisUpdate y nextUpdate
        tbs.setThisUpdate(self._thisUpdate)
        tbs.setNextUpdate(self._nextUpdate)
        ###-> Add to the crl the list of revoked certificates
        for cert in self._addToCRL:
            tbs.addRevokedCertificate(cert[0], self.crl_date_format(cert[1]))
        tbs.setRevokedCertificates()

        #Add Field "TBSCertList" to the CRL
```

```
        crl.setTbsCertList(tbs)
        #Sign the CRL
        key = EVP.load_key(self._portThread._privateKeyPath)
        key.reset_context(md='sha1')
        key.sign_init()
        message = tbs.getAsn1()
        encMessage = encoder.encode(message)
        key.sign_update(encMessage)
        sign = key.sign_final()
        finalSignature = self.bin(binascii.b2a_hex(sign))
        objSignature = _crl.Signature()
        objSignature.setSignature(finalSignature)
        #Add signature to CRL's Signature Field
        crl.setSignature(objSignature)
        crl_id = str(int(time.time()))
        #Certificate en DER format
        crl_der = encoder.encode(crl.getAsn1())
        f = open('./etc/crls/crl_' + crl_id + '.der', 'w')
        f.write(crl_der)
        f.close()
        #Certificate en PEM format
        crl_b64 = base64.b64encode(crl_der)
        aux = []
        aux.append("-----BEGIN X509 CRL-----")
        i=0
        while i < len(crl_b64):
            start = i
            end = start + 64
            aux.append(crl_b64[start:end])
            i += 64
        aux.append("-----END X509 CRL-----")
        crl_pem = '\n'.join(aux)
        f = open('./etc/crls/crl_' + crl_id + '.pem', 'w')
        f.write(crl_pem)
        f.close()
        return crl_der


    def crl_date_format(self, dateStr):
        print dateStr
        return time.strftime('%y%m%d%H%M%SZ', time.strptime(dateStr, '%Y-%m-%d %H:%M:%Sz'))

    def bin(self, a):
        s=''
        t = {'0':'0000', '1':'0001', '2':'0010', '3':'0011', '4':'0100', '5':'0101', '6':'0110', '7':'0111', '8':'1000',
'9':'1001', 'a':'1010', 'b':'1011', 'c':'1100', 'd':'1101', 'e':'1110', 'f':'1111'}
        for i in a:
            s+=t[i]
        return s

    def join(self,timeout=None):
        threading.Thread.join(self, timeout)
```

Clase CRLPortThread: /src/server/crl/CRLPortThread.py

```
#
# Creates a socket listening to an specific port and waiting for a client
# connection requesting for a CRL (Certificate Revocations List). When a
# request is received, it validates if the client is not revoked, if its
# signature is valid (just for standard mode), and then builds a
# CRL Response encoded in ASN.1 to send back to the client, containing a
# list with all revoked certificates at this moment.
#

import threading, socket, time
import crl.CacheUpdater as updater
import CRLSocketThread
import test.CRLRequestsThread as reqGen

class CRLPortThread(threading.Thread):
      IP      = 'localhost'
      MAX_CONN = 100

      def __init__(self, rsm, serverProps, dbmodule, mode):
         threading.Thread.__init__(self)
         self._listening     = True
         self._serverSocket   = None
         self._requestsThread = None #Thread for CRL Requests
         self._cacheUpdater   = None #Thread for updating Cache
         self._BANDWIDTH     = 0
         #Save a instance to the revocation servers manager
         self._rsm = rsm
         #Set startMode
         self._startMode = mode
         #Database module instance
         self._database = dbmodule
         #Read params from properties
         self._type  = serverProps.getProperty('type')
         self._class = serverProps.getProperty('class')
         self._name  = serverProps.getProperty('name')
         #Number of seconds of validity of the CRL
         self._validityPeriod   = int(serverProps.getProperty('validityPeriod'))
         #Number of CRLs to issue within a validity period
         self._overissuedFactor = int(serverProps.getProperty('overissuedFactor'))
         #Location path of Certificate
         self._certificatePath  = serverProps.getProperty('certificatePath')
         #Private Key
         self._privateKeyPath   = serverProps.getProperty('privateKeyPath')
         self._port          = int(serverProps.getProperty('port'))
         self._serverLog       = serverProps.getProperty('serverLog')
         #CRL Cache
         self._cacheCRL = None
         #Create a thread that updates the cache each VP/OF seconds. This thread also fills the cache for
the first time.
         waitTime = self._validityPeriod / self._overissuedFactor
         self._cacheUpdater = updater.CacheUpdater(self, waitTime)
         self.log("Initializing CRL Cache Updater Thread ...")
         self.log("CRL Server Initialized. Listening to port " + str(self._port))
```

```
    def run(self):
        """The CRL SCH is started. It creates a socket and listens to clients requesting for CRLs. When a
connection is received it creates a new socket between the server and the client to send the CRL."""

        #Start the thread to generate and update the CRL periodically
        self._cacheUpdater.start()
        #Start listening to Client's CRL requests
        self._serverSocket = socket.socket()
        self._serverSocket.bind((self.IP, self._port))
        self._serverSocket.listen(self.MAX_CONN)
        self._socketThreadsArray = []
        while self._listening:
            (clientsocket,address) = self._serverSocket.accept()
            socketThread = self.getSocketThreadInstance(clientsocket, address)
            self._socketThreadsArray.append(socketThread)
            socketThread.start()
            self.verifyStoppedThreads()


    def getSocketThreadInstance(self, clientSocket, address):
        """Returns a new instance of the CRLSocketThread : creates a new Thread which opens a socket
to allow communication between client and server (for sending CRLs)."""
        return CRLSocketThread.CRLSocketThread(self, clientSocket, address)

    def setListening(self, l):
        self._listening = l

    def getDataBase(self):
        return self._database

    def log(self, msg):
        """Writes logs in server log file."""
        f = open(self._serverLog, 'a')
        f.write("[" + time.asctime() + "] " + str(msg) + '\n')
        f.close()

    def close(self):
        self._serverSocket.close()

    def stop(self):
        self._listening = False
        self._cacheUpdater.stop()
        self._cacheUpdater.join()
        self.close()

    def join(self,timeout=None):
        threading.Thread.join(self, timeout)

    def verifyStoppedThreads(self):
        idx = 0
        for x in self._socketThreadsArray:
            if x.getListening() == False:
                x.join()
                self._socketThreadsArray.pop(idx)
            idx += 1
```

Clase CRLSocketThread: /src/server/crl/CRLSocketThread.py

```
#
# Class CRLSocketThread
# extends from threading.Thread
#
# A new CRLSocketThread is created when the CRL Server Socket receives a connection from a client. It receives as parameter a socket to
# allow communication between the CRL Server and the client requesting data.
# Sends to client the list of current Revoked Certificates (CRL) encoded in ASN.1 format.
#

import threading, time, hashlib
from asn1.crl import CRL
from pyasn1.codec.der import encoder, decoder
from M2Crypto import X509


class CRLSocketThread(threading.Thread):
    BYTES = 4096

    def __init__(self, portThread, socket, address):
        threading.Thread.__init__(self)
        self._clientSocket = socket
        self._portThread   = portThread
        self._address      = address
        self._listening    = True
        self._bytesRecv    = None


    def run(self):
        """Receive request, process message received, and send CRL in response if the client has a valid certificate (is not revoked)."""
        #Receive the client's request for the CRL
        self._bytesRecv = self._clientSocket.recv(self.BYTES)
        self._portThread.log("Recibo conexion de " + str(self._address[0]) + ":" + str(self._address[1]))
        self._portThread.log("Received :: " + str(len(self._bytesRecv)) + " BYTES")
        #Read the ASN.1 DER certificate of the client
        clientCertificate = X509.load_cert_string(self._bytesRecv, X509.FORMAT_DER)
        self._portThread.log("Certificate of Client requesting CRL:")
        self._portThread.log(clientCertificate.as_text())
        #Get the requesting client's serial number
        client_serial_number = clientCertificate.get_serial_number()
        self._portThread.log("Serial number of client requesting CRL:")
        self._portThread.log(client_serial_number)
        #Verify if client is revoked
        issuer_name = clientCertificate.get_issuer().as_text()
        issuer_name_hash = hashlib.md5(issuer_name).hexdigest()
        issuer_key      = clientCertificate.get_pubkey().as_der()
        issuer_key_hash = hashlib.md5(issuer_key).hexdigest()
        data = {"issuer_name_hash":issuer_name_hash, "issuer_key_hash":issuer_key_hash, "serial_number":client_serial_number}
```

```
        isRevoked = self._portThread.getDataBase().newDBSession().isRevoked(data)
        if isRevoked:
            #If the client is revoked, it cannot request any CRL
            self.portThread.log("Client with serial_number : "+str(client_serial_number) + "  cannot
request CRLs --> IS REVOKED")
        else:
            #If client is active, send him the CRL
            response = self._portThread._cacheCRL
            if response != None:
                bw = len(response)
                self._portThread._BANDWIDTH += bw
                self.addBandWidthStats()
                self._clientSocket.send(response)
                time.sleep(5)
                self._portThread._BANDWIDTH -= bw
                self.addBandWidthStats()
            else:
                self.portThread.log("Internal Error with CRL :S")
        #Close the client socket
        self._clientSocket.close()
        #Finalize thread execution
        self._listening = False


    def addBandWidthStats(self):
        """Writes bandwith stats logs in server log file."""
        statsLog = './log/bw.log'
        st = self._portThread._BANDWIDTH
        ts = time.time() - self._portThread._rsm._startTime
        f = open(statsLog, 'a')
        f.write(str(ts) + '\t' + str(st) + '\n')
        f.close()


    def getListening(self):
        return self._listening

    def join(self,timeout=None):
        threading.Thread.join(self, timeout)
```

Clase DatabaseModule: /src/server/db/DatabaseModule.py

```
import Connection
import Transaction

class DatabaseModule:

    def __init__(self):
        self.connection = None
        self.transaction = None
        self.parameters = None
```

```
    def initialize(self, parameters):
        """Creates a connection to the Database and a Transaction Object instance"""
        self.parameters = parameters


    def newDBSession(self):
        """Returns the transaction instance which allows execution of sql queries"""
        self.connection  = Connection.Connection(self.parameters).getConnection()
        return Transaction.Transaction(self.connection)
```

Clase Connection: /src/server/db/Connection.py

```
import _mysql

class Connection:
    def __init__(self, db_params):
        self.params = db_params

    def getConnection(self):
        """Returns a connection object to the postgresql DB."""
        db = _mysql.connect (user   = self.params.getProperty('user'),
                             passwd = self.params.getProperty('password'),
                             host   = self.params.getProperty('host'),
                             db     = self.params.getProperty('db'),
                             port   = int(self.params.getProperty('port')) )
        return db
```

Clase revokedCertsEntry: /src/server/db/revokedCertsEntry.py

```
class revokedCertsEntry:
    """This class represents an object modeling the table revoked_certs from DB."""
    def __init__(self, entry):
        self.issuer_name_hash     = entry[0].strip()
        self.issuer_key_hash      = entry[1].strip()
        self.serial_number        = entry[2]
        self.not_after            = entry[3].strip()
        self.revocation_time      = entry[4].strip()
        self.id_revocation_reason = entry[5]

    def getIssuer_name_hash(self):
        return self.issuer_name_hash

    def getIssuer_key_hash(self):
        return self.issuer_key_hash

    def getSerial_number(self):
        return self.serial_number
```

```
    def getNot_after(self):
        return self.not_after

    def getRevocation_time(self):
        return self.revocation_time

    def getId_revocation_reason(self):
        return self.id_revocation_reason
```

Clase Transaction: /src/server/db/Transaction.py

```
import revokedCertsEntry

class Transaction:
    """Performs different transactions on the DB. This is the only class allowed to execute SQL queries in the DB."""
    def __init__(self, db):
        self.conn = db


    def listAllRecords(self):
        """List all the revoked certificates in DB."""
        revoked_certificates = []
        try:
            self.conn.query("""SELECT * FROM revoked_certs""")
            rst = self.conn.store_result()
            regs = int(rst.num_rows())
            if regs == 0:
                return []
            for i in range(0, regs):
                entry = rst.fetch_row()[0]
                revokedCert = revokedCertsEntry.revokedCertsEntry(entry)
                revoked_certificates.append(revokedCert)
        except Exception as ex:
            print ex
            return []
        return revoked_certificates


    def getNumberOfRecords(self):
        """Counts the number of revoked certificates in DB."""
        try:
            self.conn.query("""SELECT* FROM revoked_certs""")
            rst = self.conn.store_result()
            return int(rst.num_rows())
        except Exception as ex:
            print ex
            return 0


    def isRevoked(self, params):
        """Checks in DB if the certificate specified by parameter is revoked."""
        try:
            self.conn.query("""SELECT * FROM revoked_certs WHERE issuer_name_hash = '""" +
```

```
params['issuer_name_hash'] + """' AND issuer_key_hash = '""" + params['issuer_key_hash'] + """' AND
serial_number = '""" + str(params['serial_number']) + """'""")
            rst = self.conn.store_result()
            regs = int(rst.num_rows())
            if regs > 0:
               ret = True
            else:
               ret = False
         except Exception as ex:
            ret = False
            print ex
         return ret


      def deleteAllRecords(self):
         """Delete all entries in DB."""
         try:
            self.conn.query("""DELETE FROM revoked_certs""")
            self.conn.commit()
         except Exception as ex:
            print ex
         try:
            self.cur.execute("""DELETE FROM issuers""")
            self.conn.commit()
         except Exception as ex:
            print ex


      def insertRevocationRecord(self, params):
         """Inserts a new entry in DB (for new revoked certificates)."""
         issuer = {'name_hash':params['issuer_name_hash'], 'key_hash':params['issuer_key_hash']}
         exists  = self.existsIssuer(issuer)

         if not exists:
            self.insertIssuer(issuer)
         try:
            self.conn.query("""INSERT INTO revoked_certs VALUES ('""" + params['issuer_name_hash'] +
"""', '""" + params['issuer_key_hash'] + """', '""" + str(params['serial_number']) + """', (select
CONCAT(DATE_ADD(now(), INTERVAL 3 DAY), 'z')), (select concat(now(), 'z')), """ +
str(params['id_revocation']) + """)""")
            self.conn.commit()
         except Exception as ex:
            print ex


      def deleteRevocationRecord(self, params):
         """Deletes an existing entry in DB (for expired certificates)."""
         try:
            self.conn.query("""DELETE FROM revoked_certs WHERE issuer_name_hash = '""" +
params['issuer_name_hash'] + """' AND issuer_key_hash = '""" + params['issuer_key_hash'] + """' AND
serial_number = '""" + str(params['serial_number']) + """'""")
            self.conn.commit()
         except Exception as ex:
            print ex
```

```
    def searchExpiredCertificates(self):
        """Search in DB if any of the revoked certificates has expired."""
        try:
            expiredCertificates = []
            self.conn.query("""SELECT * FROM revoked_certs WHERE not_after < now()""")
            rst = self.conn.store_result()
            regs = int(rst.num_rows())
            if regs == 0:
                return []
            for i in range(0, regs):
                entry = rst.fetch_row()[0]
                expiredCert = revokedCertsEntry.revokedCertsEntry(entry)
                expiredCertificates.append(expiredCert)
            return expiredCertificates
        except Exception as ex:
            print ex
            return []


    def existsIssuer(self, params):
        """Determines if an issuer exists in database"""
        try:
            self.conn.query("""SELECT * FROM issuers WHERE name_hash = '""" + params['name_hash'] +
""""' AND key_hash = '""" + params['key_hash'] + """'""")
            rst = self.conn.store_result()
            regs = int(rst.num_rows())
            if regs > 0:
                return True
            else:
                return False
        except Exception as ex:
            print ex
            return False


    def insertIssuer(self, params):
        """Inserts an issuer in database"""
        try:
            self.conn.query("""INSERT INTO issuers VALUES ('""" + params['name_hash'] + """', '""" +
params['key_hash'] + """')""")
            self.conn.commit()
        except Exception as ex:
            print ex
```

Clase CleanExpiredCerts: /src/server/scrp/CleanExpiredCerts.py

```
#
# Class CleanExpiredCerts
# Extends from threading.Thread
#
# Searchs actively the DB looking for expired certificate entries and delete them.
#
```

```
import threading, time

class CleanExpiredCerts(threading.Thread):

    def __init__(self, portThread, wait):
        threading.Thread.__init__(self)
        self._portThread   = portThread
        self._database     = self._portThread._database
        self._sleep        = wait
        self._isAlive      = True
        self._expiredCerts = None


    def run(self):
        while self._isAlive:
            time.sleep(self._sleep)
            self._expiredCerts = self._database.newDBSession().searchExpiredCertificates()
            self.log("Executing .... " + str(len(self._expiredCerts)) + " expired certs found!")
            for i in range(len(self._expiredCerts)):
                entry = self._expiredCerts[i]
                self._portThread.removeCertEntry(entry)


    def stop(self):
        self._isAlive = False


    def join(self,timeout=None):
        threading.Thread.join(self, timeout)


    def log(self, msg):
        logMsg = "[CleanExpiredCertsThread - "+ time.asctime() +"] " +  msg + '\n'
        f = open(self._portThread._serverLog, 'a')
        f.write(logMsg)
        f.close()
        print(logMsg)
```

Clase SCRPPortThread: /src/server/scrp/SCRPPortThread.py

```
#
# Class   SCRPPortThread
# Extends from threading.Thread
#
# Creates a socket listening to an specific port and waiting for SCRP Clients
# revocation requests. When a request is received, it validates if the client
# is active, and proceed to revoke its certificate (add to Revocation DB).
#
import threading, time, socket
import random, hashlib
import CleanExpiredCerts
import SCRPSocketThread
```

```
import test.RevocationsThread as revoc
import test.ExpirationsThread as expir
import test.ClientGenerationThread as gener
from pyasn1.codec.der import decoder
from M2Crypto import X509

class SCRPPortThread(threading.Thread):
    #Constants
    IP      = 'localhost'
    MAX_CONN = 100


    def __init__(self, rsm, serverProps, dbmodule, startmode, testProps):
        threading.Thread.__init__(self)
        #Save a instance to the revocation servers manager
        self._rsm = rsm
        #Read params from properties
        self._type              = serverProps.getProperty('type')
        self._class             = serverProps.getProperty('class')
        self._name              = serverProps.getProperty('name')
        self._revokerCertificate   = serverProps.getProperty('revokerCertificate')
        self._responderCertificate = serverProps.getProperty('responderCertificate')
        self._responderPrivateKey  = serverProps.getProperty('responderPrivateKey')
        self._serverLog         = serverProps.getProperty('serverLog')
        self._port              = int(serverProps.getProperty('port'))

        if testProps != None:
            #Test parameters
            self._issuerCert        = testProps.getProperty('issuerCertificate')
            self.populationSize     = float(self._rsm.p.general.getProperty('numberOfCertificates'))
            self.revocationPercentage  = float(testProps.getProperty('revocationPerc'))
            self.certificatesToRevoke  = float(self.populationSize * self.revocationPercentage)
            self.revokedCerts       = 0
            self.eventsRate         = int(testProps.getProperty('eventsRate'))
            self.revocationsRate    = self.eventsRate / (1 - self.revocationPercentage)
            self.expirationsRate    = self.eventsRate / self.revocationPercentage
            self.clientsGenerationRate = self.expirationsRate
            self.transitory         = True

        #Global parameters
        self.serverSocket = None
        self.listening    = True
        self.log("Creating SCRP Server --> SCRPPortThread::__init__")
        #Get startMode
        self._startMode = startmode
        self.log("Start Mode : " + str(startmode))
        #Database module
        self._database = dbmodule
        self.log("Database connection OK!")

        if self._startMode == self._rsm.STANDARD_MODE:
            print("SCRP -- STANDARD MODE")
            #Read certificates in ASN (binary)
            self.m_revokerCert   = open(self._revokerCertificate, 'r').read()
            self.m_responderCert = open(self._responderCertificate, 'r').read()
            self.m_privateKey    = open(self._responderPrivateKey, 'r').read()
            #Initialize cleanExpiredCertsThread (search and delete expired certificates from DB). If search
```

```
interval is not specified in the config file, the default value is 60.
            self.log("Initializing CleanExpiredCertsThread ...")
            self.cleanExpiredCertsThread = CleanExpiredCerts.CleanExpiredCerts(self,
int(serverProps.getProperty("cleanExpiredCertsThread", 60)))
        elif self._startMode == self._rsm.TEST_MODE:
            print("SCRP -- TEST MODE")
            self.m_issuerCert    = open(self._issuerCert ,'r').read()
            #Issuer name and key hash ::> the certificate of the issuer of clients in TEST_MODE is defined
in variable self.m_issuerCert (DER format)
            #Get from this issuer the key_hash and name_hash
            asn1IssuerCert       = decoder.decode( self.m_issuerCert )[0]
            issuerCert           = X509.load_cert_string(self.m_issuerCert, X509.FORMAT_DER)
            issuer_name          = issuerCert.get_issuer().as_text()
            self.issuer_name_hash = hashlib.md5(issuer_name).hexdigest()
            issuer_key           = issuerCert.get_pubkey().as_der()
            self.issuer_key_hash  = hashlib.md5(issuer_key).hexdigest()
            self.log("ISSUER NAME HASH : " + self.issuer_name_hash)
            self.log("ISSUER KEY HASH  : " + self.issuer_key_hash)
            #Initiliaze threads for expirations and revocations
            self.revocationsThread = revoc.RevocationsThread(self)
            self.expirationsThread = expir.ExpirationsThread(self)
            self.clientGenThread   = gener.ClientGenerationThread(self)


    def log(self, msg):
        """Writes logs in server log file."""
        f = open(self._serverLog, 'a')
        _log = "[" + time.asctime() + "] " + str(msg) + '\n'
        f.write(_log)
        f.close()
        print (_log)


    def removeCertEntry(self, entry):
        """Deletes a revoked certificates from DB (used for Expired Certificates)."""
        _entry = {"issuer_name_hash":entry.getIssuer_name_hash(),
"issuer_key_hash":entry.getIssuer_key_hash(), "serial_number":entry.getSerial_number()}
        self._database.newDBSession().deleteRevocationRecord(_entry)
        self.log("Deleting expired cert from database --> w/serial number : " +
str(_entry['serial_number']))


    def revokeCert(self, entry):
        """Inserts a new Revoked Certificate in DB. Notify to all SCHs the new revocation entry."""
        #Insert entry in DB
        self._database.newDBSession().insertRevocationRecord(entry)
        self.log("Certificate w/serial number : " + str(entry['serial_number'])+ " inserted in DB!")
        #Stats:: Number of revoked certs has changed
        self._rsm._stats[1].statsLog()


    def expireCert(self, idx, entry):
        """Removes expired Certificate from the array of clients, and deletes it from DB if it was also
revoked."""
        #Delete from database if certificate is Revoked
        if entry != None:
```

```
            self.log("Deleting expired cert from database --> w/serial number : " +
str(entry['serial_number']))
            self._database.newDBSession().deleteRevocationRecord(entry)
            self.certificatesToRevoke += 1
        #Remove this client from rsm's clients array
        self._rsm._testClients.pop(idx)
        self._rsm._numberOfCertificates -= 1
        print("Current Number of Clients : " + str(self._rsm._numberOfCertificates))
        #Update populationCerts Stats
        self._rsm._stats[0].statsLog(self._rsm._numberOfCertificates)
        self._rsm._stats[1].statsLog()


    def verifyStoppedThreads(self):
        idx = 0
        for x in self._socketThreadsArray:
            if x.getListening() == False:
                x.join()
                self._socketThreadsArray.pop(idx)
            idx += 1


    def run(self):
        """We begin the simulation. Depending on the application start mode, we create a socket to
listen for client's requests, or we wait until system is stable, to start revocations and expirations
Thread."""
        print("OPENING SERVER SOCKET :: LISTENING SCRP REQUESTS TO PORT " + str(self._port))
        #If we are in STANDARD-MODE : creates a socket to listen to clients Requests
        if self._startMode == self._rsm.STANDARD_MODE:
            #Starts cleanExpiredCerts Thread!
            print("Starting CleanExpiredCerts Thread!")
            self.cleanExpiredCertsThread.start();
            self.serverSocket = socket.socket()
            self.serverSocket.bind((self.IP, self._port))
            self.serverSocket.listen(self.MAX_CONN)
            self._socketThreadsArray = []
            while self.listening:
                (clientsocket,address) = self.serverSocket.accept()
                socketThread = self.getSocketThreadInstance(clientsocket, address)
                self._socketThreadsArray.append(socketThread)
                socketThread.start()
                self.verifyStoppedThreads()
        #If we are in TEST-MODE : revokes certificates until system is stable --> revokedCerts =
population x revocationProbability(p)
        elif self._startMode == self._rsm.TEST_MODE:
            print('certificates to revoke : ' + str(self.certificatesToRevoke))
            #Once all clients has been created (N = populationSize), we start revoking certificates
randomly until : r = p x N
            while self.certificatesToRevoke > 0 and self.transitory:
                tryToRevoke = random.sample(self._rsm._testClients, 1)[0]
                data = {"issuer_name_hash":self.issuer_name_hash,
"issuer_key_hash":self.issuer_key_hash, "serial_number":tryToRevoke.serial_number,
"id_revocation":0}
                isRevoked = self.getDataBase().newDBSession().isRevoked(data)
                if isRevoked == False:
                    self.revokeCert(data)
                    self.certificatesToRevoke -= 1
```

```
                else:
                    self.log("Certificate w/serial number " + str(tryToRevoke.serial_number) + "is already
revoked!")
            #At this point, the system is stable, and the simulation may go on.
            #So, we start revocationsThread, expirationsThread and clientGenerationsThread...
            self.transitory = False
            self.revocationsThread.start()
            self.expirationsThread.start()
            self.clientGenThread.start()


        def getSocketThreadInstance(self, clientSocketInstance, address):
            """Returns a new instance of the SCRPSocketThread : creates a new Thread which opens a
socket to allow communication between client and server."""
            return SCRPSocketThread.SCRPSocketThread(clientSocketInstance, address, self)


        def setListening(self, l):
            self.listening = l


        def getDataBase(self):
            return self._database


        def close(self):
            self.serverSocket.close()


        def stop(self):
            self.listening = False
            if self._startMode == 1:
                self.revocationsThread.stop()
                self.revocationsThread.join()
                self.expirationsThread.stop()
                self.expirationsThread.join()
                self.clientGenThread.stop()
                self.clientGenThread.join()
            elif self._startMode == 0:
                self.cleanExpiredCertsThread.stop()
                self.cleanExpiredCertsThread.join()
            self.close()


        def join(self,timeout=None):
            threading.Thread.join(self, timeout)
```

Clase SCRPSocketThread: /src/server/scrp/SCRPSocketThread.py

```
# Class SCRPSocketThread
# extends from threading.Thread
#
```

```
# Its use is only intended for STANDARD-MODE simulations.
# Receives a SCRP Request sent form a SCRP client. This modules processes this request, check status in the DB and send the result
# of the revocation request in a SCRP Response format.

import threading, hashlib, binascii
from M2Crypto import EVP, X509
from pyasn1.type import univ
from pyasn1.codec.der import encoder, decoder
import asn1.scrp.SCRPResponse as scrp_resp
import asn1.x509.X509Certificate as x509_cert

class SCRPSocketThread(threading.Thread):
    _BYTES       = 4096
    OID_SHA1withRSA = '1.2.840.113549.1.1.5'

    def __init__(self, socket, address, portThread):
        threading.Thread.__init__(self)
        self._listening    = True
        self._clientSocket = socket
        self._portThread   = portThread
        self._address      = address
        self.m_responderCert = None
        self.m_revokerCert   = None
        self.m_privateKey    = None
        self._bytesRecv    = None
        self._isRevoked    = False
        self._hasSignature = False
        self._authorized   = False
        self._privateKeyPath  = self._portThread._responderPrivateKey
        self.signature       = None
        self.certToRev       = None
        self.revocationReason = 0
        self.revocationTime   = None
        self.issuer_key_hash  = ''
        self.issuer_name_hash = ''


    def run(self):
        if (self._listening == True):
            #Receive the request revocation
            self._bytesRecv = self._clientSocket.recv(self._BYTES)
            self._portThread.log("Recibo conexion de " + str(self._address[0]) + ":" + str(self._address[1]))
            self._portThread.log("Received :: " + str(len(self._bytesRecv)) + " BYTES")
            #Process the request received
            result = self.processSCRPRequest()
            #Create the response
            response = self.createResponse(result)
            #Send response of revocation to the client
            self._clientSocket.send(encoder.encode(response))
            #Close the socket
            self._clientSocket.close()
            #End thread execution
            self.stop()


    def stop(self):
```

```
        self._listening = False


    def processSCRPRequest(self):
        SUCCESFUL     = 0
        ALREADYREVOKED = 1
        INTERNALERROR  = 2
        SIGREQUIRED    = 3
        UNAUTHORIZED   = 4
        try:
            #Decode the request in ASN.1 DER format
            request = decoder.decode( self._bytesRecv )[0]
            #Get  tbsRequest field and digital signature
            tbsRequest = request.getComponentByPosition(0)
            self.signature  = request.getComponentByPosition(1).getComponentByPosition(1)
            #Data about the revocation process
            self._certToRev         = tbsRequest.getComponentByPosition(0)
            self._certificateToRevoke = X509.load_cert_string(encoder.encode(self._certToRev),
X509.FORMAT_DER)
            self._issuerCertificate   = tbsRequest.getComponentByPosition(1)
            self._issuerCertificate   = X509.load_cert_string(encoder.encode(self._issuerCertificate),
X509.FORMAT_DER)
            self._revocationTime = str(tbsRequest.getComponentByPosition(2))
            self._revocationReason = int(tbsRequest.getComponentByPosition(3))
            self._portThread.log("Certificate to Revoke :")
            self._portThread.log(self._certificateToRevoke.as_text())
            self._portThread.log("Issuer Certificate :")
            self._portThread.log(self._issuerCertificate.as_text())
            self._portThread.log("Revocation Time :")
            self._portThread.log(self._revocationTime)
            self._portThread.log("Revocation Reason :")
            self._portThread.log(self._revocationReason)
            #Serialnumber of the certificate to revoke
            self._serialNumberToRev = self._certificateToRevoke.get_serial_number()
            self._portThread.log("Serial number of certificate to revoke :")
            self._portThread.log(self._serialNumberToRev)
            #Generate Issuer Name and Public Key Hash
            issuer_name = self._issuerCertificate.get_issuer().as_text()
            self.issuer_name_hash = hashlib.md5(issuer_name).hexdigest()
            issuer_key  = self._issuerCertificate.get_pubkey().as_der()
            self.issuer_key_hash = hashlib.md5(issuer_key).hexdigest()
            self._portThread.log("ISSUER NAME HASH : " + self.issuer_name_hash)
            self._portThread.log("ISSUER KEY HASH  : " + self.issuer_key_hash)
            #Is the certificate already revoked?
            data = {"issuer_name_hash":self.issuer_name_hash, "issuer_key_hash":self.issuer_key_hash,
"serial_number":self._serialNumberToRev}
            isRevoked = self._portThread.getDataBase().newDBSession().isRevoked(data)
            if isRevoked:
                self._isRevoked = True
                self._portThread.log("THE CERTIFICATE IS ALREADY REVOKED!!!!!")
                return ALREADYREVOKED
            else:
                #If it is not already revoked, we proceed to revoke
                if self.signature:
                    self._hasSignature = True
                    self._authorized   = True
```

```
                entry = {"issuer_name_hash":self.issuer_name_hash,
"issuer_key_hash":self.issuer_key_hash, "serial_number":self._serialNumberToRev,
"id_revocation":self._revocationReason}
                self._portThread.revokeCert(entry)
                return SUCCESFUL
            else:
                return SIGREQUIRED
        except Exception as ex:
            self._portThread.log("An internal exception/error has ocurred")
            print (ex)
            return INTERNALERROR


    def createResponse(self, result):
        #Create TBSResponse field
        tbsResponse = scrp_resp.Response()
        tbsResponse.setCertToRev(self._certToRev)
        tbsResponse.setResult(result)
        #Create Signature Field
        #-->Set Algorithm Identifier
        signatureAlgorithm = scrp_resp.AlgorithmIdentifier()
        signatureAlgorithm.setAlgorithm(self.OID_SHA1withRSA)
        signatureAlgorithm.setParameters(univ.Null(''))
        signature = scrp_resp.Signature()
        signature.setSignatureAlgorithm(signatureAlgorithm.getAsn1Spec())
        #-->Sign the response
        privKey = open(self._privateKeyPath).read()
        message = tbsResponse.getAsn1Spec()
        key = EVP.load_key_string(privKey)
        key.reset_context(md='sha1')
        key.sign_init()
        msg = encoder.encode(message)
        key.sign_update(msg)
        sign = key.sign_final()
        certSignature = self.bin(binascii.b2a_hex(sign))
        signature.setSignature(certSignature)
        #Create the SCRPResponse
        scrpResponse = scrp_resp.SCRPResponse()
        scrpResponse.setTbsResponse(tbsResponse.getAsn1Spec())
        scrpResponse.setSignature(signature.getAsn1Spec())
        asn1 = scrpResponse.getAsn1Spec()
        return asn1


    def bin(self, a):
        s = ''
        t = {'0':'0000', '1':'0001', '2':'0010', '3':'0011', '4':'0100', '5':'0101', '6':'0110', '7':'0111', '8':'1000',
'9':'1001', 'a':'1010', 'b':'1011', 'c':'1100', 'd':'1101', 'e':'1110', 'f':'1111'}
        for i in a:
            s += t[i]
        return s


    def join(self,timeout=None):
        threading.Thread.join(self, timeout)
```

```
    def getListening(self):
        return self._listening
```

Clase Api:
/src/server/stats/Api.py

```
import socket, threading
from pyasn1.type import char
from pyasn1.codec.der import encoder, decoder

class Api(threading.Thread):
    IP   = 'localhost'
    PORT = 8001
    MAX_CONN = 100

    def __init__(self, rsm):
        threading.Thread.__init__(self)
        self._rsm        = rsm
        self.serverSocket = None
        self.listening    = True
        self._allThreads  = []


    def run(self):
        self.serverSocket = socket.socket()
        self.serverSocket.bind((self.IP, self.PORT))
        self.serverSocket.listen(self.MAX_CONN)
        while self.listening:
            (clientsocket,address) = self.serverSocket.accept()
            threadInstance = Handler(self._rsm, clientsocket, address)
            self._allThreads.append(threadInstance)
            threadInstance.start()
            self.verifyStoppedThreads()


    def verifyStoppedThreads(self):
        idx = 0
        for x in self._allThreads:
            if x.getRunning() == False:
                x.join()
                self._allThreads.pop(idx)
            idx += 1


    def stop(self):
        self.listening = False


    def join(self,timeout=None):
        threading.Thread.join(self, timeout)
```

```
class Handler(threading.Thread):
    def __init__(self, rsm, _socket, _address):
        threading.Thread.__init__(self)
        self._rsm     = rsm
        self._socket  = _socket
        self._address = _address
        self._running = True


    def run(self):
        dataReceived = self._socket.recv(128)
        #Total of Certificates (C)
        c = self._rsm._numberOfCertificates
        #Revoked Certficartes (V)
        v = self._rsm.dbModule.newDBSession().getNumberOfRecords()
        #For the perspective of clients:
        #Good Certificates (G) :: G = C - V
        #Operative Certs      :: O = C - K
        #Unknown revoked Certs :: U = V - K
        response = char.PrintableString(str(c) + "," + str(v))
        print "STATS A ENVIAR : "
        print response
        self._socket.send(encoder.encode(response))
        self._socket.close()
        self._running = False


    def join(self,timeout=None):
        threading.Thread.join(self, timeout)


    def getRunning(self):
        return self._running
```

Clase PopulationSize, RevokedCertsNumber: /src/server/stats/Stats.py

```
import time, threading

class PopulationSize(threading.Thread):
    def __init__(self, rsm):
        threading.Thread.__init__(self)
        self._rsm = rsm
        self._generateStats = True


    def run(self):
        while self._generateStats:
            self.statsLog(self._rsm._numberOfCertificates)
            #Check number of active certificates each 60 seconds
            time.sleep(60)


    def statsLog(self, log):
```

```
        f = open('./log/populationSize.log', 'a')
        f.write(str(self.getCurrentSimulationTime()) + "\t" + str(log) + "\n")
        f.close()


    def getCurrentSimulationTime(self):
        return  time.time() - self._rsm._startTime


    def stop(self):
        self._generateStats = False


    def join(self,timeout=None):
        threading.Thread.join(self, timeout)



class RevokedCertsNumber():

    def __init__(self, rsm):
        self._rsm = rsm


    def statsLog(self):
        revokedCertsNum = self._rsm.dbModule.newDBSession().getNumberOfRecords()
        f = open('./log/revokedCertsNumber.log', 'a')
        f.write(str(self.getCurrentSimulationTime()) + "\t" + str(revokedCertsNum) + "\n")
        f.close()


    def getCurrentSimulationTime(self):
        return  time.time() - self._rsm._startTime
```

Clase ExponencialRandomGenerator: /src/server/utils/randomVariable.py

```
import random, math

class ExponencialRandomGenerator():
    num   = 0
    X     = 0
    media = 0

    def __init__(self, mean):
        self.media = mean
        self.num   = 0
        self.X     = 0


    def getValue(self):
        self.num = random.random()
        self.X = (-1)*self.media*(math.log1p(self.num-1))
```

```
        self.X = math.floor(self.X*10)/10
        return self.X
```

Clase EndSimulation: /src/server/test/EndSimulation.py

```
import time, threading

class EndSimulation(threading.Thread):
    def __init__(self, rsm, endtime):
        threading.Thread.__init__(self)
        self._rsm = rsm;
        self._endTime = endtime

    def run(self):
        time.sleep(self._endTime);
        self._rsm.exit_program()
```

Clase RevocationsThread: /src/server/test/RevocationsThread.py

```
#
# Class RevocationsThread
# extends from threading.Thread
#
# Thread executed every X seconds ,where X is the revocations Rate : eventsRate / 1 -
RevokedCertProbability(p)
# Revokes a random certificate from the array of clients (population)
#
import threading, time, random
import utils.randomVariable as generator

class RevocationsThread(threading.Thread):
    def __init__(self, portThread):
        threading.Thread.__init__(self)
        #Save SCRPPortThread instance
        self._portThread = portThread
        #The time between revocvation request arrivals is readed from SCRPPortThread (eventsRate/ 1 -
p)
        self._timeBetweenArrivals = self._portThread.revocationsRate
        #Initialize Random Generator : Random Variable Exponential (mean = _timeBetweenArrivals)
        self._expRandom = generator.ExponencialRandomGenerator(self._timeBetweenArrivals)
        self._generateRevocations = True


    def run(self):
        """Each time the thread is executed, tries to revoke randomly one certificate, and then sleeps N
seconds (N : Exponential Random Variable) until next execution."""
        while self._generateRevocations:
            #Wait "_timeBetweenArrivals" seconds...
```

```
            time.sleep(self._expRandom.getValue())
            #Revoke a Certificate
            revokedCertsNum = self._portThread._rsm.dbModule.newDBSession().getNumberOfRecords()
            revokedRatio = float(revokedCertsNum)/float(self._portThread._rsm._numberOfCertificates)
            if revokedRatio < self._portThread.revocationPercentage:
               self.revokeCertificate()


      def revokeCertificate(self):
         """Selects randomly one certificate from the array of clients. Verify if this certificate is no already
revoked. Finally, it revokes the Certificate."""
         certToRevoke = random.sample(self._portThread._rsm._testClients, 1)[0]
         data = {"issuer_name_hash":self._portThread.issuer_name_hash,
"issuer_key_hash":self._portThread.issuer_key_hash, "serial_number":certToRevoke.serial_number,
"id_revocation":0}
         isRevoked = self._portThread.getDataBase().newDBSession().isRevoked(data)
         if not isRevoked:
            self._portThread.log("RevocationsThread : Trying to revoke certificate w/serial number " +
str(certToRevoke.serial_number))
            self._portThread.revokeCert(data)
         else:
            self._portThread.log("Certificate w/serial number " + str(certToRevoke.serial_number) + " is
already revoked!")


      def stop(self):
         """Stops Generate Revocations Thread."""
         self._generateRevocations = False


      def join(self,timeout=None):
         threading.Thread.join(self, timeout)
```

Clase ExpirationsThread: /src/server/test/ExpirationsThread.py

```
#
# Class ExpirationsThread
# extends from threading.Thread
#
# Thread executed every X seconds , where X is the expirations Rate :
eventsRate/revocationProbability(p)
# Expires a random certificate from the array of clients (population). If it was revoked, it also removes it
from database.
#
import random, threading, time
import utils.randomVariable as generator

class ExpirationsThread(threading.Thread):
      def __init__(self, portThread):
         threading.Thread.__init__(self)
         #Save SCRPPortThread instance
         self._portThread = portThread
         #The time between revocvation request arrivals is read from SCRPPortThread (eventsRate / p)
```

```
        self._timeBetweenArrivals = self._portThread.expirationsRate
        #Initialize Random Generator : Random Variable Exponential (mean = _timeBetweenArrivals)
        self._expRandom = generator.ExponencialRandomGenerator(self._timeBetweenArrivals)
        self._generateExpirations = True


    def run(self):
        """Each time this thread is executed, expires randomly one certificate and then sleeps N seconds
(N: Exponential Random Variable) until next execution."""
        while self._generateExpirations:
            #Wait "_timeBetweenArrivals" seconds...
            time.sleep(self._expRandom.getValue())
            #Expire a Certificate
            self.expireCertificate()


    def expireCertificate(self):
        """Selects randomly one certificates from the array of clients (population) and expires it. If it was
revoked, it also deletes it from DB."""
        index       = random.randint(0, len(self._portThread._rsm._testClients)-1)
        certToExpire = self._portThread._rsm._testClients[index]
        data = {"issuer_name_hash":self._portThread.issuer_name_hash,
"issuer_key_hash":self._portThread.issuer_key_hash, "serial_number":certToExpire.serial_number,
"id_revocation":0}
        isRevoked = self._portThread.getDataBase().newDBSession().isRevoked(data)
        if isRevoked:
            self._portThread.log("Revoked Certificate w/serial number " + str(certToExpire.serial_number)
+ " has expired! ... Deleting from database!")
            self._portThread.expireCert(idx=index, entry=data)
        else:
            self._portThread.log("Active Certificate w/serial number " + str(certToExpire.serial_number) +
" has expired!")
            self._portThread.expireCert(idx=index, entry=None)


    def stop(self):
        """Stops Generate Revocations Thread."""
        self._generateExpirations = False


    def join(self,timeout=None):
        threading.Thread.join(self, timeout)
```

Clase ClientGenerationThread: /src/server/test/ClientGenerationThread.py

```
#
# Class ClientGenerationThread
# extends from threading.Thread
#
# Thread executed every X seconds , where X is the generations Rate :
eventsRate/revocationProbability(p) = expirationsRate
# Creates a new client and adds it to the rsm's clients Array.
#
```

```
import threading, time
import utils.randomVariable as generator

class ClientGenerationThread(threading.Thread):
    def __init__(self, portThread):
        threading.Thread.__init__(self)
        self._portThread = portThread
        self._timeBetweenArrivals = self._portThread.clientsGenerationRate
        self._expRandom = generator.ExponencialRandomGenerator(self._timeBetweenArrivals)
        self._generateClients = True


    def run(self):
        """Each time this thread is executed, creates a new Client and then sleeps N seconds (N :
Exponential Random Variable) until next execution."""
        while self._generateClients:
            time.sleep(self._expRandom.getValue())
            self.createClient()


    def createClient(self):
        """Calls rsm's function generateClient() to create a new Client."""
        self._portThread._rsm.generateClient()


    def stop(self):
        self._generateClients = False


    def join(self,timeout=None):
        threading.Thread.join(self, timeout)
```

Clase TestClient: /src/server/test/TestClient.py

```
from M2Crypto import X509

class TestClient:
    DEFAULT_CERT = './certs/public/default.pem'

    def __init__(self, serial_number):
        self.certificate = X509.load_cert_string(open(self.DEFAULT_CERT).read(), X509.FORMAT_PEM)
        self.certificate.set_serial_number(long(serial_number))
        self.serial_number = serial_number
        self.isRevoked = False
```

REFERENCIAS

[1] Cooper, et al. Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. IETF RFC 5208, May 2008. http://tools.ietf.org/html/rfc5280

[2] J. Tapia. *Design and Evaluation of Certificate Revocation Systems*. Universitat Politècnica de Catalunya, Department de'Enginyeria Telemàtica (2004).

[3] W. Polk, W. Ford and D. Solo. *Internet X.509 Public Key Infraestructure Certificate and Certificate Revocation List (CRL) Profile*, 2002, RFC 3280.

[4] R. Rivest. The MD5 Message-Digest Algorithm. IETF RFC 1321, April 1999. http://www.rfc-es.org/rfc/rfc1321-es.txt

[5] P. Jones.US Secure Hash Algorithm (SHA1). IETF RFC 3174, September 2008. http://tools.ietf.org/html/rfc5280

[6] J. Muñoz, O. Esparza, C. Gañán, J.Parra-Arnau, *PKIX Certificate Status in Hybrid MANETs.* Universitat Politècnica de Catalunya, Department Enginyeria Telemàtica (2009).

[7] Domingo Aladrén, M.C, "*Diferenciación de servicios y mejora de la supervivencia en redes ad-hoc conectadas a redes fijas*", Tesis Doctoral, Universitat Politècnica de Catalunya, Castelldefels, 2005.

[8] F. Bai, A. Helmy, *A Survey of Mobility Models in Wireless Adhoc Networks,* University of Southern California, USA.

[9] V. Cerf, S. Burleigh, A. Hooke, L. Torgerson, R. Durst, K. Scott, H. Weiss. *Delay-Tolerant Network Architecture,* DTN Research Group Internet Draft, Draft 2, March 2003.

[10] Kevin Fall, *A Delay-Tolerant Network Architecture for Challenged Internets*. Intel Research Berkeley, Technical Report IRB-TR-03-003.

[11] *XGEN: Una breve introducción a ASN.1 y BER*. Microsoft Support. Junio 2006.

[12] ITU-T Recommendation X.690. *ASN.1 Encoding Rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rues (DER)*, 1995.

[13] L. Cassel, R. Austing. *Computer Networks and Open Systems. An Application Development Perspective.* ISBN 0-7637-1122-5. Jones Bartlett Publisher, 2000.

[14] M. Myers, R. Ankney, A. Malpani, S. Galperin, C.Adams. X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP. IETF RFC 2560, June 1999. http://tools.ietf.org/html/rfc2560.txt

[15] C. Gañán, J.Muñoz, O.Esparza, J.Mata-Díaz, J.Alins. *A user-based mechanism for issuing certifícate status information in disruption tolerant networks.* Universitat Politècnica de Catalunya. 2010.

[16] M. Raya, D. Jungels, P. Papadimitratos, I. Aad, JP. Hubaux. *Certificate Revocation in Vehicular Networks.* School of Computer and communicaction Sciences, EPFL Switzerland. LCA-Report-2006-006.

[17] J. Arnau. *Nuevo criterio para la estimacion de información de estado de certificados en MANET*. Universitat Politècnica de Catalunya, Department de'Enginyeria Telemàtica (2009).