Autoridad Certificante

Ejercicio N°3

Objetivos	● Diseño y construcción de sistemas con acceso distribuido ● Encapsulación de Threads y Sockets en Clases ● Definición de protocolos de comunicación ● Protección de los recursos compartidos ● Uso de buenas prácticas de programación en C++
Instancias de Entrega	**Entrega 1:** clase 8 (16/04/2019). **Entrega 2:** clase 10 (30/04/2019).
Temas de Repaso	● Definición de clases en C++ ● Contenedores de STL ● Excepciones / RAII ● Move Semantics ● Sockets ● Threads
Criterios de Evaluación	● Criterios de ejercicios anteriores ● Eficiencia del protocolo de comunicaciones definido ● Control de paquetes completos en el envío y recepción por Sockets ● Atención de varios clientes de forma simultánea ● Eliminación de clientes desconectados de forma controlada

Índice

Introducción

Se deberá desarrollar un sistema reducido de generador de certificados digitales. Permitirá a los usuarios de las aplicaciones cliente crear o revocar un certificado a un servidor único.

Descripción

Una autoridad certificante es una entidad de confianza, dedicada a la generación, validación, renovación y revocación de certificados.

Estos servicios son principalmente utilizados para garantizar la integridad y autenticidad de las comunicaciones digitales vía el protocolo TLS (Transport Layer Security), utilizados en las comunicaciones web (HTTPS), las comunicaciones de correos electrónicos (SMTP. POP3, IMAP) o para el resguardo de documentos digitales (firma digital).

El prototipo de sistema deberá poder solamente crear un certificado y revocarlo usando algoritmos *simples* (no-criptográficamente seguros) con rango de valores numéricos pequeños (pocos bits y por lo tanto, inseguros).

Certificado

Un certificado es un documento de texto, con datos del usuario solicitante (como su nombre, región, dominio, y sus claves públicas de encriptación), y datos de la autoridad que lo validó (como su nombre y firma digital que lo autentica).

El certificado cuenta con algunos metadatos extra como pueden ser, por ejemplo, los rango de fechas de validez del mismo.

Cuando se habla de firma digital se refiere a una operación matemática que se le realiza al documento, de forma tal que se logre una huella que genere autenticidad de la información (la información provino de quien se supone que debía venir) y su integridad (los datos del archivo no fueron alterados de ninguna forma posible). Para esto se utilizan algoritmos de *Hashing* y *encriptación asimétrica*.

A los efectos del trabajo, se espera que la autoridad genere un certificado con la siguiente estructura:

```
certificate:
        serial number: 11 (0x0000000b)
        subject: Federico Manuel Gomez Peter
        issuer: Taller de programacion 1
        validity:
                not before: Mar 28 21:33:04 2019
                not after: May 27 21:33:04 2019
        subject public key info:
                modulus: 253 (0x00fd)
                exponent: 17 (0x11)
```

Serial number

Un id autoincremental **sin signo de 4 bytes**. Su representación se hará de forma decimal y de forma hexadecimal.

Subject

Representa la información del cliente que solicita el certificado. Para los alcances del trabajo, solo será el nombre del cliente.

Issuer

Información relevante de la autoridad certificante. Para este trabajo, será un string fijo, con el nombre de la materia.

Validity

Rango de fechas de validez del certificado en cuestión. Se compone de los atributos *not before*, con la fecha de inicio del certificado, y del atributo *not after*, con la fecha de fin de validez del archivo. La fecha se representará de la forma:

MMM DD HH:mm:SS YYYY

Este dato puede venir del archivo de configuración del cliente, que se detallará más adelante. Si así no lo fuera, el cliente enviará la fecha actual como el inicio del certificado, y como fecha de fin de validez será 30 días posteriores a la fecha actual.

Para esto se **recomienda revisar** las bibliotecas ***ctime*** e ***iomanip***[1].

Subject public key info

Información de la clave pública del cliente. Esta se compone del *módulo*, un **número sin signo de 2 bytes** de extensión (su valor se representa tanto decimal como hexadecimalmente), y el *exponente*, **un número sin signo de 1 byte**.

Todos estos atributos se representan de forma jerárquica, en el orden impuesto en el ejemplo anterior, y donde cada nodo hijo del padre se encuentra desplazado (indentado) a la derecha con un **tab**.

Mostrando los tabs y saltos de línea como ‘`\t`‘ y ‘`\n`‘ respectivamente, el certificado sería:

certificate:\n\tserial number: 11 (0x0000000b)\n\tsubject: Federico Manuel Gomez Peter\n\tissuer: Taller de programacion 1\n\tvalidity:\n\t\tnot before: Mar 28 21:33:04 2019\n\t\tnot after: May 27 21:33:04 2019\n\tsubject public key info:\n\t\tmodulus: 253 (0x00fd)\n\t\texponent: 17 (0x11)

Notese ademas como el archivo no termina con ningún fin de línea.

Hash

La función de hash aplicará la sumatoria del valor ASCII de todos los caracteres que componen el archivo en un **entero sin signo de 2 bytes**.

Por ejemplo si el archivo contuviera el string “hola” (sin ningún fin de línea), su hash sería igual a 420. Cada caracter del archivo es visto como un **número sin signo de 1 byte** (104, 111, 108, 97) y es sumado en un **contador de 2 bytes también sin signo**.

En el certificado ejemplo visto en la sección anterior, el hash resultante sería (en su representación decimal):

20603

RSA

RSA[2] es un algoritmo de encriptación asimétrico que hace uso de la operación modular y de la exponenciación para la encriptación y desencriptación.

En este algoritmo, tanto la encriptación como la desencriptación usan el mismo procedimiento. Este consta de tomar elementos de **1 byte sin signo,** elevarlos a una potencia y aplicarle la operación *módulo.*

Para esto se necesitan tres datos: el exponente público (1 byte sin signo), el exponente privado (1 byte sin signo), y el módulo (entero de 2 bytes sin signo). Las claves se componen de la siguiente forma:

Clave pública: (exponente público, módulo)
Clave privada: (exponente privado, módulo)

El pseudocódigo de este algoritmo es:

```
/*
 * El retorno es un entero de 4 bytes sin signo.
 */
ret := 0
/*
 * Recorro los dos bytes del hash calculado previamente
 */
for i from 0 to 1:
   byte := (hash >> (i * 8)) and 0xff

   // elevo el valor de character al exponente público de la clave
   result := 1
   for j from 1 to exponent:
     result := (result * byte) mod module

   ret := ret + (byte << (i * 8))

endfor
return ret
```

Siguiendo el ejemplo anterior del mensaje “hola”, el cifrador recibirá el hash calculado previamente, de valor 420 (`0x01a4` en su representación hexadecimal). Suponiendo que se quiere cifrar con la siguiente clave:

(19, 253)

Se toma el primer byte del hash (`0xa4` o `164` en su valor decimal). Elevado a la 19 vale:

`164^19 = 1207905928726874669932385907137927167606784`

Aplicando módulo se obtiene que:

`164^19 mod 253 = 98 = 0x62`

Se incrementa el valor del entero de retorno (que ahora valdrá 98) y se repite el procedimiento para el segundo byte del hash:

`1^19 mod 253 = 1`

Y para sumarlo al entero de retorno, debo aplicar un shifteo a izquierda de 8 posiciones (que equivale a multiplicar por 256 al entero). El valor de la encriptación final vale, entonces, 354 (`0x0162`).

Formato de Línea de Comandos

Servidor

```
./server <puerto/servicio> <claves> <índice>
```

Donde `<puerto/servicio>` es el puerto TCP (o servicio) en donde estará escuchando la conexiones entrantes.

El parámetro `<claves>` es el nombre del archivo que contiene el la información de las claves privadas y públicas de la autoridad certificante. El servidor deberá leerlo al inicio y obtener la información relevante. Los datos de las claves se representan en un archivo de texto con números separados por uno o más espacios en el siguiente orden:

1 - Exponente público, un número que se puede representar como un entero de **1 byte sin signo.**
2 - Exponente privado, un número de **1 byte sin signo.**
3 - Modulo, un número de **2 bytes sin signo.**

Un ejemplo de este archivo es el siguiente:

```
13 17 253
```

Por último, el parámetro `<índice>` es la ruta al archivo donde se listarán los distintos **subjects** registrados en el sistema, con sus respectivas claves públicas, separados por una nueva línea (**\n**). En la primer línea de este archivo se guardará el próximo número de serie a insertar. Un ejemplo del mismo puede ser:

```
5
Federico Manuel Gomez Peter; 19 253
Ezequiel Werner; 17 253
Matias Lafroce; 13 253
Martín Di Paola; 139 253
```

En donde el siguiente certificado a crear tendrá el número de serie 5, y los certificados creados fueron para Los distintos subjects de cada linea.

Se **recomienda** investigar la libreria ***fstream***, en particular el ***operador >>***.

Cliente

El cliente tendrá dos modos de uso:

new

```
./client <ip/hostname> <puerto/servicio> new <claves clientes> <pública servidor> <información certificado>
```

El cliente se conectará al servidor corriendo en la máquina con dirección IP `<ip>` (o `<hostname>`), en el puerto (o servicio) TCP `<puerto/servicio>`.

En este modo, se ejecutará el comando de creación (**new**) del certificado. La `<información certificado>` es la dirección del archivo con la información que requiere el servidor para crear dicho archivo:

1 - El subject (nombre del cliente)
2 - Fecha de inicio (en el formato descrito en secciones anteriores)

3 - Fecha de fin (en el formato descrito en secciones anteriores)

Los últimos dos parámetros son opcionales. En el caso de que estos no se encuentren, la fecha de inicio debe ser el momento actual de ejecución, y la fecha de fin, exactamente 30 días después. Estos parámetros se encuentran separados por un carácter de *nueva línea*. Por ejemplo:

```
Federico Manuel Gomez Peter
Mar 28 19:00:24 2019
May 27 19:00:24 2019
```

Finalmente, `<claves clientes>` apunta al archivo con la información de las claves públicas y privadas del cliente, y `<pública servidor>` contiene la clave pública del servidor.

El formato de ambos es idéntico al descrito en el archivo de las claves del servidor, con la salvedad de que el archivo `<pública servidor>` tendrá dos números: el exponente público y el módulo.

revoke

```
./client <ip/hostname> <puerto/servicio> revoke <claves clientes> <pública servidor> <certificado>
```

El cliente se conectará, y enviará a la autoridad certificante el certificado del archivo `<certificado>`. El servidor validará debidamente dicho certificado y, si la validación fuese correcta, elimina al subject de su memoria.

Códigos de Retorno

El servidor devolverá 0 si su ejecución fue exitosa, o 1 en caso contrario. El cliente deberá devolver siempre 0, imprimiendo por salida estándar los mensajes de errores que se detallan en la sección siguiente.

Entrada y Salida Estándar

Servidor

El servidor no imprimirá nada por salida estándar.
En cambio, esperará el caracter `‘q’` por entrada estándar. Cuando lo reciba, el servidor deberá finalizar todas las conexiones y cerrarse.

Cliente

Por salida estándar imprimirá datos de la huella y del hash.

Cuando se ejecuta el comando **new**, se imprimirá primero la huella recibida del servidor, en la siguiente línea la huella desencriptada, y en la siguiente línea la huella calculada por el cliente, de la siguiente forma:

```
Huella del servidor: <Huella recibida>
Hash del servidor: <Huella desencriptada del servidor>
Hash calculado: <Hash calculado del certificado recibido>
```

En el caso de que se ejecute el comando de revocación. Se debe imprimir la siguiente información:

```
Hash calculado: <Hash calculado del certificado a revocar>
Hash encriptado con la clave prívada: <Hash encriptado con la clave pública del cliente>
Huella enviada: <Hash cifrado>
```

Seguido a esto, se imprimirá también por la **salida estándar** los siguientes mensajes de error (si los hubiera):

Imprimirá `"Error: argumentos invalidos.\n"` si:
- El modo a ejecutar es distinto de `new` o `revoke`
- Se reciben un número incorrecto de parámetros.

Imprimirá `"Error: usuario no registrado.\n"` si:
- El subject enviado al servidor para revocar (`revoke`) no se encuentra registrado.

Imprimirá `"Error: ya existe un certificado.\n"` si:
- El subject tiene un certificado creado anteriormente.

Imprimirá `"Error: los hashes no coinciden.\n"` si:
- El hash recibido por el servidor no coincide por el hash calculado en el cliente, cuando se quiere hacer una alta del certificado.
- El hash recibido por el cliente no coincide por el hash calculado en el servidor, cuando se quiere dar una baja del certificado.

Protocolo de comunicación

Formato del mensaje

Estos son los delineamientos generales:

- Todas las cantidades y longitudes se enviarán en enteros de 4 bytes sin signo en big endian.
- Todos los strings, incluidos los archivos, se enviarán ***sin*** el delimitador `'\0'`, enviando primero (prefijo) la longitud del string.
- Los comandos o acciones a realizar así como también los códigos de retorno se enviarán en enteros de 1 bytes sin signo.
- El módulo de una clave se envía como un entero de 2 bytes sin signo en big endian.
- El exponente de una clave se envía como un entero de 1 byte sin signo.
- La huella del certificado se envía como un entero de 4 bytes sin signo en big endian.

- El certificado se debe enviar pasando los valores de los distintos atributos en el orden que aparecen en el archivo (primero se envía el valor del número de serie, luego el subject, siguiendo por el valor del issuer, la fecha de inicio, la fecha de finalización, el módulo del cliente y luego el exponente).

new

El cliente quiere crear (`new`) un nuevo certificado. Para eso le envía al servidor el siguiente mensaje:

- El comando o acción **new** codificado con el valor numérico **0**. En otras palabras, envía el número **0**.
- A continuación se envía el subject. Como se indicó en la sección **Formato del mensaje**, para enviar un string se debe enviar primero su longitud (4 bytes, sin signo, big endian) y luego el contenido del string sin el `'\0'`.
- Luego se envía el módulo del cliente como un entero sin signo de 2 bytes.
- Se envía el exponente como un entero sin signo de 1 byte.
- La fecha de inicio del certificado.
- La fecha de finalización de la validez del certificado.

El servidor deberá responder con un código de retorno (1 byte, sin signo) con un valor numérico igual a **0** si el subject ya tiene un certificado vigente o **1** en caso contrario. Si el certificado se crea correctamente, almacena en memoria el subject y la clave pública.

Si el certificado ya existe, el cliente finaliza con el mensaje de error detallado en la sección correspondiente. En cambio, si no existe, el cliente debe recibir los siguientes elementos:

- El certificado creado por el servidor.
- El hash del certificado, encriptado con la clave privada del servidor y luego encriptado con la clave pública del cliente.

El cliente debe debe desencriptar el hash, primero usando su propia clave privada, y luego usando la clave pública del servidor. Seguidamente calcula el hash del certificado recibido y compara si este cálculo coincide con el del que recibió del servidor. Si así lo fuera, guarda el certificado en un archivo, notifica al servidor su correcta recepción (enviando un entero de **1 byte sin signo** de valor **0**) y termina su ejecución normalmente. Si los hashes no coinciden, debe notificar al servidor con un entero **de 1 byte sin signo** de valor **1**, para que el servidor elimine de su almacenamiento interno los datos del certificado creado recientemente. Inmediatamente el cliente cierra su ejecución escribiendo en pantalla el mensaje descripto en la sección anterior.

revoke

El cliente quiere dar de baja su certificado (revoke). Para esto, el cliente enviará:

- El comando o acción **revoke** codificado con el valor numérico **1**. En otras palabras, envía el número **1**.
- Luego envía el certificado a revocar.
- Finalmente, envía el hash, encriptado primero con la clave privada del cliente, y luego con la pública del servidor.

El servidor debe primero validar que quien envió esta solicitud de baja sea quien dice ser. Para esto, obtiene del certificado recibido el subject. Con este dato observa si existe registrado en su mapa un usuario con ese nombre. Si así no fuese, devuelve un código de retorno de **1** en formato **1 byte sin signo.** Si existe un subject registrado, desencripta, primero con la clave privada del servidor y la pública del cliente almacenada

en memoria, la huella recibida. Luego compara el hash recibido con el hash que aplica el servidor al certificado recibido. Si estos no coinciden, devuelve el número **2** en formato **1 byte sin signo.** Si coinciden, el servidor validó la autoría del solicitante, lo elimina a este de su mapa y le responde al cliente con un código **0.**

El cliente cierra normalmente (o imprimiendo los mensajes de error correspondientes si así lo amerita) e imprime por salida estándar el hash calculado y la huella enviada al servidor en el formato descripto en la sección anterior.

Formato de archivos

Índice

El servidor tendrá su índice guardado en un archivo. Este será leído al momento de inicializarse y será guardado (actualizado) al momento de cerrarse. El índice contiene información sobre el próximo número de serie a utilizar al crear un certificado, y además todos los sujetos con un certificado vigente (que no hayan sido revocados), y sus respectivas claves públicas.

El formato del archivo consta de una línea con el número de serie y cero o más líneas con los subjects registrados. Cada línea de un subject se encuentra compuesta por el nombre del cliente registrado, el caracter punto y coma (;), un espacio, el exponente público del cliente, otro espacio y finalemente el módulo del cliente. Un ejemplo de un índice será el siguiente:

```
5
Federico Manuel Gomez Peter; 19 253
Ezequiel Werner; 17 253
Matias Lafroce; 13 253
Martín Di Paola; 139 253
```

Información del certificado

Cuando el cliente solicite crear un nuevo certificado, se le debe pasar por parámetro un archivo que contenga la información necesaria para generarlo. Esta información consta de los siguientes atributos:

- El subject
- La fecha de inicio de validez del certificado
- La fecha de expiración de validez del certificado

Estos campos deben presentarse separados por un carácter de nueva línea. Los campos de fechas de inicio y expiración son opcionales. En el caso de que estos no estuvieran, el cliente debe crear estos campos internamente. La fecha de inicio será el momento actual en el que este se encuentre ejecutando, y la fecha de fin será exactamente 30 días posteriores a la fecha actual.

El siguiente contenido representa un ejemplo de la información del usuario a generar el certificado (con las fechas incluidas explícitamente):

```
Federico Manuel Gomez Peter
Mar 28 19:00:24 2019
May 27 19:00:24 2019
```

Archivo de claves

Las distintas claves que requieren ambos sistemas se les brindará por medio de distintos archivos que respetan la siguiente estructura:

```
<exponente público> <exponente privado> <módulo>
```

Un ejemplo de este archivo es el siguiente:

```
13 17 253
```

Vale la aclaración de que los distintos campos pueden estar separados por 1 o más espacios, únicamente se garantiza que estos se encuentran en una misma línea. En el caso particular de las claves que se le brindan al cliente, cuando se le pase las claves del servidor, solo tendrá los datos del exponente público y del módulo.

Ejemplos de Ejecución

Se procede a presentar unos ejemplos de ejecución explicando los distintos pasos:

Creación del certificado

El servidor se invoca de la siguiente forma:

```
./server 2000 server.keys index.txt
```

El servidor escuchará clientes por el puerto 2000. El archivo *server.keys* contiene las claves públicas y privadas del servidor:

```
13 17 253
```

Es decir, la clave pública del servidor será el par (13, 253), donde 13 es el exponente público y 253 el módulo. La clave privada será (17, 253). El archivo index.txt es el estado de la CA antes de su cierre anterior. En este ejemplo será la primera vez que se levante el servidor, por lo que el archivo será inexistente (lo creará el server al momento de guardar su estado).

Una vez ejecutado el servidor, se levanta un cliente con los siguientes comandos:

```
./client localhost 2000 new certificate.req client.keys serverPublic.keys
```

El cliente se conectará a la ip localhost, en el puerto 2000, y ejecutará el comando **new**. El archivo *certificate.req* contiene la información necesaria para hacer el request al servidor, es decir, tendrá el subject y el período temporal de validez del certificado. En este ejemplo se brindó la siguiente información:

```
Federico Manuel Gomez Peter
Mar 28 21:33:04 2019
May 27 21:33:04 2019
```

El request consiste de un certificado para el **subject** Federico Manuel Gomez Peter, **válido a partir del 28/03/2019 hasta el 27/05/2019**. Las claves del cliente son:

```
19 139 253
```

Y el archivo de claves públicas del servidor tiene sólo la clave pública de este:

```
13 253
```

El cliente procede a notificarle al servidor el deseo de crear un certificado nuevo, enviando el **comando 0 en formato de 1 byte sin signo.** Luego envía el subject, la clave pública del cliente y el periodo de validez. El servidor crea el certificado (no lo guarda en ningún archivo), **si y solo si no existe ningún subject registrado previamente.** El certificado creado exitosamente posee la siguiente estructura:

```
certificate:
        serial number: 1 (0x00000001)
        subject: Federico Manuel Gomez Peter
        issuer: Taller de programacion 1
        validity:
                not before: Mar 28 21:33:04 2019
                not after: May 27 21:33:04 2019
        subject public key info:
                modulus: 253 (0x00fd)
                exponent: 19 (0x13)
```

Guarda en memoria el subject y la clave pública del cliente registrado recientemente. Seguidamente envía el comando de ejecución exitosa (**0 en formato de 1 byte sin signo**) y calcula el hash del certificado, cuyo valor es:

```
20509
```

A este valor se lo encripta, primero aplicando la clave privada del servidor, donde se obtiene:

```
39039
```

A este se lo encripta nuevamente, aplicando la clave pública del cliente:

```
38011
```

Este valor se envía al cliente. Este valida que lo recibido proviene efectivamente de la autoridad certificante. Para esto, calcula el hash del certificado y desencripta la huella proveniente del servidor (primero con la clave privada del cliente y luego con la pública del servidor). En el caso de que estos valores coincidan, se guarda el certificado en un archivo cuyo path es `<subject>.cert`. En el caso del ejemplo, el archivo que se crea es `Federico Manuel Gomez Peter.cert`. Por salida estándar imprimirá el mensaje descripto en la sección correspondiente, que en caso de este ejemplo será:

```
Huella del servidor: 38011
Hash del servidor: 20509
Hash calculado: 20509
```

Y para finalizar notifica al servidor de su correcta recepción, enviandole un entero de **1 byte sin signo** de valor **0**.

Revocación

El servidor se invoca de la siguiente forma:

```
./server 2000 server.keys index.txt
```

La clave será la misma que en el ejemplo anterior. Continuando con el ejemplo anterior, el archivo `index.txt` tendrá la siguiente información:

```
2
Federico Manuel Gomez Peter; 19 253
```

El cliente se levanta con los siguientes argumentos

```
./client localhost 2000 revoke “Federico Manuel Gomez Peter.cert” client.keys
                                serverPublic.keys
```

Las claves son las mismas que en el ejemplo anterior, y el certificado es el creado anteriormente. El cliente procede a notificarle al servidor el deseo de dar de baja un certificado, enviando el **comando 1 en formato de 1 byte sin signo.** Luego envía el certificado y la huella de este (el hash encriptado con la clave privada del cliente y luego con la pública del servidor). Luego de pasar por la doble encriptación, imprime por salida estándar los datos de la huella:

```
Hash calculado: 20509
Hash encriptado con la clave prívada: 49014
Huella enviada: 5313
```

Esto se envía al servidor. El servidor luego de recibir el certificado obtiene de él el campo subject que usará para fijarse si tiene un certificado registrado en memoria. Si no tuviese ningún certificado registrado con ese subject, el servidor se lo notifica al cliente enviando el valor **1** como un **entero sin signo de 1 byte.** Si existiese un certificado registrado para dicho subject, el servidor obtiene la clave pública que tiene almacenada en memoria. Con esta clave desencripta la huella recibida y, luego de calcular el hash con el certificado recibido, lo compara con esta. Si ambos hash, el calculado por el cliente y el calculado por el servidor, no coinciden, envía el valor **2**. Si coinciden, elimina los datos almacenados del subject y retorna el valor **0**.

Restricciones

La siguiente es una lista de restricciones técnicas exigidas por el cliente:

1. El sistema debe desarrollarse en ISO C++11.
2. Está prohibido el uso de variables globales.
3. La búsqueda que el servidor hace para buscar sujetos registrados se debe realizar en un tiempo menor que N.
4. Debe haber uso de polimorfismo.
5. Sobrecargar el **operador >>** y el **operador <<** para alguna clase.

Referencias

[1]: https://en.cppreference.com/w/cpp/chrono/c/localtime
[2]: https://cs.uns.edu.ar/~ldm/mypage/data/ss/info/ejemplo-rsa.pdf