

# Uso del API de cifrado de JAVA (JCA/JCE)

SSI 2016/17

21 de septiembre de 2016

## Índice

<b>1. Objetivos</b>	<b>1</b>
<b>2. Descripción</b>	<b>1</b>
2.1. Simplificaciones . . . . .	1
2.2. Requisitos . . . . .	2
<b>3. Desarrollo</b>	<b>3</b>
3.1. Actores . . . . .	3
3.2. Programas a desarrollar . . . . .	3
<b>4. Entrega</b>	<b>5</b>
<b>5. Herramientas a utilizar</b>	<b>5</b>

## 1. Objetivos

- Poner en práctica los conocimientos adquiridos respecto a algoritmos criptográficos
- Conocer y utilizar un API estándar para el desarrollo de aplicaciones criptográficas de complejidad media
  - En este caso se hará uso del API *Java Cryptography Architecture (JCA)* y del provider (implementación de JCA) BouncyCastle.

## 2. Descripción

Se trata de desarrollar una colección de herramientas para dar soporte a la validación de las "*Compostelas*" de los peregrinos del Camino de Santiago.

Los peregrinos podrán generar su *Compostela Virtual* que será sellada por los albergues por donde pase. Finalmente, la *Oficina del Peregrino* podrá validar dicha *Compostela Virtual* para extraer los datos del peregrino y validar la autenticidad de los "sellos" emitidos por los albergues.

### 2.1. Simplificaciones

Dado que se trata de una aplicación "de juguete" se asumirán una serie de simplificaciones.

- Cada uno de los participantes (PEREGRINO, ALBERGUE, OFICINA DEL PEREGRINO) podrá generar sus propios pares de claves privada y pública que se almacenarán en ficheros (no se considera una protección del fichero con la clave privada, como sí ocurriría en una aplicación real)
- No se contemplan los mecanismos de distribución de claves públicas. Se asumirá que todas las claves públicas necesarias estarán en poder del usuario que las necesite (PEREGRINO, ALBERGUE u OFICINA DEL PEREGRINO) de forma confiable (en una aplicación real se haría uso de certificados digitales y de una *autoridad de certificación* común)
  - El PEREGRINO dispondrá de un fichero con la clave pública de la OFICINA DEL PEREGRINO
  - Cada ALBERGUE dispondrá de un fichero con la clave pública de la OFICINA DEL PEREGRINO y también tendrá acceso al fichero con la clave pública del PEREGRINO que desee "sellar" su *Compostela Virtual*.
  - La OFICINA DEL PEREGRINO contará con la clave pública (almacenada en su respectivo fichero) de cada uno de los ALBERGUES, así como con la clave pública del PEREGRINO para el cual se vaya a realizar la validación de su *Compostela Virtual*.
- Las distintas piezas de información que aporte cada participante (PEREGRINO o ALBERGUE) tendrán (antes del cifrado/firma y después del descifrado) la forma de *Strings* en formato JSON con codificación UTF8.
- La *Compostela Virtual* se materializará en un "paquete" que contendrá toda la información que vayan incorporando los distintos participantes implicados: el PEREGRINO que la generó y los ALBERGUES que la hayan "sellado"

**Nota:**

- Se aporta código para la gestión de "Paquetes" con múltiples partes codificadas en bloques de caracteres imprimibles empleando codificación Base64 (ver Codificación Base64)
- **NECESARIAMENTE** se **deberá** emplear el código proporcionado o proporcionar una implementación propia equivalente en el caso de desarrollar la prácticas en otros lenguajes diferentes a Java.
- También se aporta código adicional para simplificar la generación y parseo de cadenas en un formato JSON simplificado (sólo almacena un nivel de pares de cadenas clave-valor).
- No se contempla un almacenamiento "físico" realista de la *Compostela Virtual*, sólo se trata de implementar los programas para generar, sellar y validar la *Compostela Virtual* conforme a las especificaciones descritas en este documento.
- Al validar la *Compostela Virtual*, si todas las comprobaciones son correctas, se mostrará a la OFICINA DEL PEREGRINO los datos aportados por el PEREGRINO y los datos incorporados en cada uno de los "sellos" estampados por cada ALBERGUE que haya procesado dicha *Compostela Virtual*. En caso contrario se indicarán las comprobaciones que no hayan sido satisfactorias.
- No se contemplan comprobaciones adicionales más allá de las anteriores, como puede ser verificar el orden correcto de los "sellados" o la imposibilidad física/temporal de los "sellados" realizados.

Se asume que este tipo de comprobaciones "lógicas" las hará el personal de la OFICINA DEL PEREGRINO

## 2.2. Requisitos

Requisitos básicos a cumplir por el esquema criptográfico propuesto:

- R1.** Asegurar la **confidencialidad** del contenido (nombre, DNI, etc) incluido en la *Compostela Virtual* por parte del PEREGRINO (sólo la *oficina del peregrino* podrá tener acceso a estos contenidos).
- Opcionalmente se puede asegurar también la confidencialidad de la información (fecha, lugar, etc) incluida en los "sellos" de los ALBERGUES (sólo la OFICINA DEL PEREGRINO podría tener acceso a estos contenidos), aunque no es imprescindible.
- Para la funcionalidad requerida en este proyecto basta con el "sellado" y esos datos pueden dejarse en claro en el "paquete" que da soporte físico a la *Compostela Virtual*.
- R2.** Garantizar que la OFICINA DEL PEREGRINO tenga la **seguridad** de que el PEREGRINO que presenta la *Compostela Virtual* es quien dice ser.

- R3. Asegurar** que el contenido del "paquete" con la *Compostela Virtual* (datos del PEREGRINO y sellos de los ALBERGUES) que se ha recibido **no** haya sido **modificado**.
- R4.** Asegurar que ni el PEREGRINO y los ALBERGUES podrán **repudiar** el contenido incluido por ellos en la *Compostela Virtual*
- R5. Asegurar** que la OFICINA DEL PEREGRINO **no** podrá realizar **cambios** en el contenido de la *Compostela Virtual* que ha recibido.
- R6.** Contar con un mecanismo mediante el cuál los ALBERGUES puedan **garantizar la fecha** y el lugar en que fue "sellada" la *Compostela Virtual* generada por un determinado PEREGRINO.
- Se pretende que esta vinculación entre *compostela virtual* y "sello" pueda ser validada por la OFICINA DEL PEREGRINO y que no pueda ser falsificada ni por el PEREGRINO, ni por otros ALBERGUES, ni por la propia OFICINA DEL PEREGRINO
- **Nota:** En este caso los ALBERGUES funcionarán de un modo parecido a las **autoridades de sellado de tiempo** de las infraestructuras de clave pública (ver más en Sellado de tiempo [wikipedia]).
- R7.** Asegurar un **coste computacional reducido** en la creación, sellado y validación de la *Compostela Virtual* **minimizando** el uso de **criptografía asimétrica**

### 3. Desarrollo

En primer lugar se deberán de analizar los requisitos anteriores, para determinar qué estrategias seguir para conseguir cada uno de ellos.

Se debe decidir qué acciones realizar en el origen (PEREGRINO), qué tareas realizarán las "*autoridades de sellado de tiempo*" (ALBERGUES) y qué comprobaciones se llevarán a cabo en el destino (OFICINA DEL PEREGRINO), además de decidir qué algoritmos concretos se emplearán.

#### 3.1. Actores

**Peregrino.** podrá generar su propio par de claves (pública y privada) y será responsable de **generar** su *Compostela Virtual* con los datos que le sean requeridos.

**Albergue.** podrá generar su propio par de claves (pública y privada) y será responsable de **sellar** la *Compostela Virtual* de un PEREGRINO dado, aportando los datos que le sean requeridos.

**Oficina del peregrino.** podrá generar su propio par de claves (pública y privada) y será responsable de extraer los datos aportados por el PEREGRINO de su propia *Compostela Virtual* y validar la información de los "sellos" emitidos por los correspondientes ALBERGUES

#### 3.2. Programas a desarrollar

Una vez decidido cómo garantizar los requisitos exigidos, el resultado final será **obligatoriamente** el desarrollo de 4 ejecutables:

- `java -cp [...] GenerarClaves <identificador>` (ya está **implementado**)
  - Usado para generar los pares de claves de los participantes: PEREGRINO, ALBERGUE y OFICINA DEL PEREGRINO
  - Se le pasa como argumento de línea de comando un identificador que se usará para componer los nombre de los archivos que se generarán
  - Genera dos ficheros: `''identificador.publica''` e `''identificador.privada''`, conteniendo las claves pública y privada de ese usuario



## 4. Entrega

- Práctica **individual** o en **parejas**.
- **Documentación a entregar:**

**Memoria** Contendrá al menos los siguiente puntos

- Descripción breve de la práctica
- Descripción y justificación de las estrategias criptográficas empleadas para asegurar los requisitos básicos exigidos, junto con los pasos que se siguen en el empaquetado, sellado y desempaquetado
- Descripción del formato/estructura del "paquete" resultante
- Descripción breve de la implementación: clases y métodos más importantes.
- Instrucciones de compilación y ejemplos de uso.
- Breve comentario sobre las simplificaciones realizadas (apartado 2.1) y sus consecuencias en una aplicación real.
- Resultados obtenidos y conclusiones

**Código fuente + casos de prueba** • **Aviso importante:**

Salvo razones debidamente justificadas en la documentación (uso de otros lenguajes de programación o similares), los ejecutables desarrollados deben de seguir las restricciones respecto a nombre, parámetros de línea de comandos y orden de los mismos especificadas en la sección 3.2 ("Programas a desarrollar"). Del mismo modo, los datos aportados por peregrinos y albergues han de almacenarse en el "Paquete" creado y presentarse por pantalla en la notación JSON simplificada gestionada por el código base aportado.

- **Fecha de entrega:** por determinar (última semana de octubre, primera semana de noviembre)  
Entrega en FAITIC o "en mano" (documentación impresa + CD ó USB con los ficheros para copiar)
- **Defensa:** esta previsto que alrededor de un 10 %-15 % de las prácticas entregadas en tiempo y forma serán seleccionadas para una pequeña defensa (5-10 min.) ante el profesor
  - fundamentalmente serán prácticas con dudas de funcionamiento o implementación, dudas de autoria o elegidas al azar
  - la defensa se realizará bien en horario de laboratorio o en horario de tutorías (se acordará con cada grupo)
- **Evaluación:**
  - Documentación correcta + empaquetado básico (sólo datos del *peregrino*) y validación/descifrado por *oficina del peregrino*: hasta 60 %
  - Adición de generación y validación de "sellos" de los *albergues*: hasta 100 %

## 5. Herramientas a utilizar

La práctica se implementará en Java utilizando el API de criptografía JCA y el provider Bouncy Castle. Se podrá realizar tanto en Windows como en Linux.

- API Java Cryptography Architecture (JCA) [antes JCE (Java Cryptography Extension)]
- Paquete de criptografía Bouncy Castle (JCA provider)
- Tutorial del API JCA (Java Cryptography Architecture) [pdf]

Código de partida:

- `Paquete.java`: encapsula un paquete formado por varios bloques. Cada bloque tiene un nombre (de tipo `String`) y un contenido (de tipo `byte[]`). La clase provee de métodos para añadir, listar y eliminar los bloques que conforman el paquete.

- `Bloque.java`: encapsula cada uno de los bloques que forman parte de un paquete.
- `PaqueteDAO.java`: métodos estáticos que dan soporte a las operaciones de lectura y escritura de paquetes empleando un formato similar al PGP *ASCII armor* (ver PGP) que usa la codificación Base64 para representar datos binarios mediante caracteres imprimibles.
  - Métodos públicos estáticos: `leerPaquete(...)` y `escribirPaquete(...)`
  - Ver en el método `main()` ejemplos de escritura y lectura de Paquetes.
  - Esta clase usa los objetos `org.bouncycastle.util.encoders.Base64` de la librería BouncyCastle para la conversión a/desde Base64, por lo que para compilar y ejecutar es necesario incluir en el CLASSPATH el correspondiente paquete JAR.
- `JSONUtils.java`: métodos estáticos con utilidades para el formateo y parseo del formato JSON simplificado usado como soporte para el almacenamiento de los datos aportados por peregrinos y albergues.
  - Métodos públicos estáticos: `json2map(...)` y `map2json(...)`
  - Ver en el método `main()` ejemplos de conversión de JSON a `Map<String,String>` y viceversa.
- `GenerarClaves.java`: genera un par de claves RSA de 512 bits