Edicom TimeStamping Authority

Manual de usuario

Título del documento:	Edicom TimeStamping Authority
Nombre del fichero:	ES EDICOM TSA Manual de usuario.odt
Versión:	2.0
Estado:	VIGENTE
Fecha:	11/02/2009
Autor:	Francisco Santonja

Revisión, Aprobación		
Revisado por:	Francisco Santonja	Fecha: 11/02/2009
Aprobado por:	Francisco Santonja	Fecha: 11/02/2009

Historial de cambios			
Versión	Fecha	Descripción de la acción	Páginas
1.0	11/02/2009	Versión Inicial	All.
2.0	16/04/2015	Versión revisada. El entorno de administración se realiza desde el Ediwin Cripto Server por lo que esa sección se elimina de este manual. Se añaden ejemplo de implementación JAVA y las referencias al mensaje Validate Request. Se elimina información superflua para el usuario final.	

Índice de contenido

1 EDICOM TSA

1.1 INTRODUCCIÓN

El propósito de este manual es describir las funcionalidades para la administración de la entidad de sellado de tiempo, quedando fuera de su alcance aspectos como políticas de seguridad y arquitectura. Para facilitar el proceso de administración también se incluye una descripción del proceso de sellado de tiempo y definiciones básicas.

> **Importante**: La administración y la gestión de usuarios se realiza actualmente a través del panel de control Web del ECS (Edicom Cripto Server). Esa información no está contemplada en este documento.

1.2 DEFINICIONES

Para los propósitos del presente documento, se aplican los siguientes términos y definiciones:

- **Autoridad de Sellado de Tiempo**: Sistema de emisión y gestión de sellos de tiempo seguros
- **Subscriptor**: Persona o entidad que solicita los servicios proporcionados por la Autoridad de Sellado de Tiempo.
- **Token de sello de tiempo**: Dispositivo de datos empleado en un proceso de creación de firma electrónica, que une la representación de un dato a un tiempo concreto, estableciendo así evidencia de que el dato existía antes de ese tiempo.
- **Usuario**: Destinatario de un Token de sello de tiempo y que confía en el mismo.

3.2. ABREVIATURAS

TSA: Autoridad de Sellado de Tiempo

TSS: Servicio de sellado de tiempo

TSQ: Solicitud de sello de tiempo

ACEDICOM: Autoridad de Certificación de EDICOM

TST: Token de sello de tiempo

IETF: Internet Engineering Task Force

CEN: Comité Europeo de Normalización

CWA: Cen Workshop Agreement

RFC: Request for comment

UTC: Universal Time Coordinated

CRL: Certificate Revocation List

FIPS: Federal Information Processing Standards

HSM: Hardware Security Module

GPS: Global Positioning System

1.3 REFERENCIAS

[1] RFC-3161 “Internet X.509 Public Key Infrastructure – Time Stamp Protocol (TSP)”

[2] ETSI TS 102 023 “Policy Requirements for time-stamping authorities”

[3] ETSI TS 101 861 “Time Stamping Profile”

[4] ETSI SR 002 176 “Algorithms and Parameters for Secure Electronic Signatures”

[5] ISO/IEC 10118-3: "Information technology - Security techniques – Hash functions - Part 3: Dedicated hash-functions".

[6] FIPS Publication 180-1 (1995): "Secure Hash Standard"

2 CONCEPTOS GENERALES

2.1 SERVICIO DE SELLADO DE TIEMPO

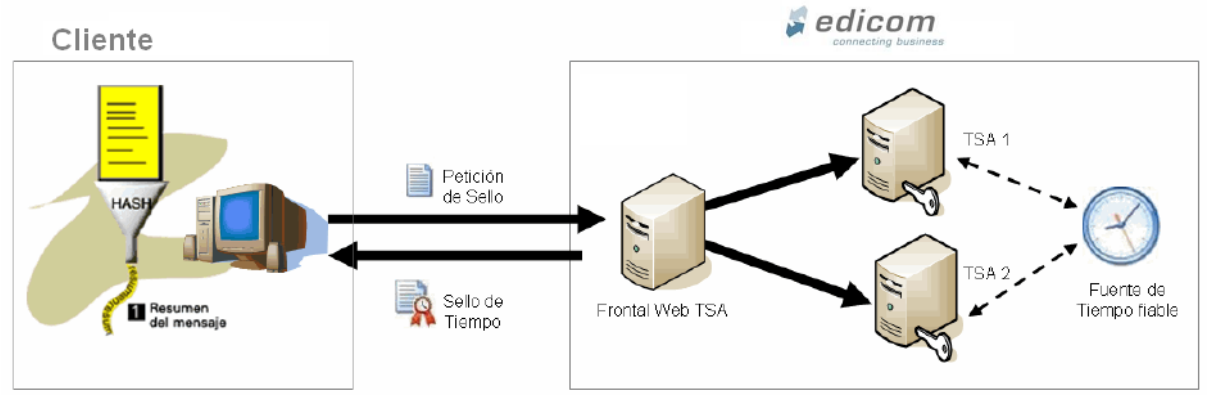
El sellado de tiempo (Timestamping) es un mecanismo on-line que permite demostrar que una serie de datos han existido y no han sido alterados desde un instante específico en el tiempo.

La implementación de la política de sellado de tiempo se debe cumplir con el protocolo definido en la norma **RFC 3161 "Internet X.509 Public Key Infrastructure Time-Stamp Protocol (TSP)".**

Los pasos para generar un sello de tiempo son los siguientes:

- El cliente calcula el hash del documento a sellar
- El cliente envía una solicitud de sello de tiempo a una URL determinada de ACEDICOM siguiendo el protocolo RFC 3161, incluyendo el hash del documento a sellar
- ACEDICOM recibe la petición, revisa si la petición está completa y correcta y realiza un control de acceso en función del usuario y password del cliente.
- Si el resultado es correcto, la TSA firma la petición generando un Sello de Tiempo (incluyendo el hash del documento, la fecha y hora obtenida de una fuente fiable y la firma electrónica de la TSA).
- El sello de tiempo se envía de vuelta al Cliente
- El Cliente debe validar la firma del sello y custodiarlo debidamente

La TSA de la ACEDICOM también mantendrá un registro de los sellos emitidos para su futura verificación si así se ha contratado con el cliente.



2.2 AUTORIDAD DE SELLADO DE TIEMPO (TSA)

La autoridad en la que confían los usuarios de los servicios de sellado de tiempo (suscriptores y las distintas partes confiantes) para la emisión de los sellos de tiempo.

2.3 SUBSCRIPTORES

Los subscriptores de este servicio son los usuarios del sistema con los que se haya suscrito el correspondiente convenio de prestación de servicios de sellado de tiempo electrónico.

Los clientes envían peticiones de sellado y reciben sellos de tiempo siguiendo el protocolo RFC3161 Time Stamp Protocol (TSP).

Los clientes deben adaptar sus sistemas para poder realizar peticiones de sellado de tiempo. Existen librerías públicas que implantan el protocolo TSP en diversos lenguajes de programación:

- BouncyCastle (http://www.bouncycastle.org): Conjunto de librerías criptográficas que implementan el protocolo TSP en los lenguajes Java y C#
- OpenTSA (http://www.opentsa.org): Es una ampliación de la librería criptográfica OpenSSL que implementa el protocolo TSP en lenguaje C.
- Digistamp (http://digistamp.com/toolkitDoc/MSToolKit.htm): Toolkit basado en la librería criptográfica CryptoAPI de Microsoft que implementa el protocolo TSP en Visual Basic
- IAIK: Incluye librerías criptográficas en Java que implementan el protocolo TSP. Estas librerías son gratuitas únicamente para propósitos no comerciales
- Adobe Reader: La aplicación Adobe Reader 8 permite validar sellos de tiempo incluidos en documentos PDF.

3 IMPLEMENTACIÓN DE LA SOLICITUD Y RESPUESTA DE SELLOS DE TIEMPO.

Las solicitudes y respuestas de sellos se se adherirán a la sintaxis de la especificación "***RFC3161 Time Stamp Protocol (TSP)***" descrito en el Apartado 3.4. "Time-Stamp Protocol via http" de la especificación, con las restricciones de la norma ETSI TS 101 861.

3.1 PROTOCOLO TIMESTAMP VÍA HTTP

La autoridad de sellado de tiempo permite peticiones de sellado tiempo tanto a traves de HTTP y HTTPS con autenticación básica.

La URL del servicio de Sellado de Tiempo de ACEDICOM es

```
http://tsa.acedicom.edicomgroup.com:9026/

https://tsa.acedicom.edicomgroup.com:9027/
```

Existen dos tipos de mensajes: los que el cliente envía a la TSA, y los que envía la TSA al cliente. Todos los mensajes están representados en notación ASN.1

Formato de la petición

```
Content-Type: application/timestamp-query
     <<the ASN.1 DER-encoded Time-Stamp Request message>>
```

Formato de la respuesta

```
Content-Type: application/timestamp-reply
     <<the ASN.1 DER-encoded Time-Stamp Response message>>
```

3.2 SOLICITUD DE SELLADO TIMESTAMP REQUEST

Este mensaje lo utiliza la entidad que quiere un sello de tiempo (solicitante) para acceder al servicio que ofrece una TSA. Tiene el siguiente formato:

```
TimeStampReq ::= SEQUENCE  {
version                 INTEGER  { v1(1) },
messageImprint          MessageImprint,
reqPolicy               TSAPolicyId              OPTIONAL,
nonce                   INTEGER                  OPTIONAL,
certReq                 BOOLEAN                  DEFAULT FALSE,
extensions              [0] IMPLICIT Extensions  OPTIONAL
```

}

Campo	Descripción
version	Versión de la petición TimeStamp (v1)
messageImprint	OID del algoritmo hash y el valor del hash de los datos
reqPolicy	OID de la política de la TSA Indica a la TSA la política bajo la cuál quiere que se proporcione el sello
Nonce	Si se incluye el **nonce** permite al cliente comprobar el retardo en la respuesta cuando no se dispone de reloj local. La respuesta debe contener este mismo número o se rechazará. El **nonce** es un número aleatorio con una elevada probabilidad de que el cliente lo genere una única vez (entero de 64 bits).
CertReq	Si el campo **certReq** está presente y con valor true, la clave publica de la TSA debe estar referenciada por el identificador ESSCertID dentro de un atributo SigningCertificate de la estructura SignedData en la respuesta. Ese campo además puede contener otros certificados. Si falta el campo **certReq** o tiene valor false entonces, el campo SigningCertificate de la estructura SignedData no debe aparecer en la respuesta.
extensions	Es una forma de permitir añadir nuevos campos en el futuro. Si se incluye algún campo de extensión que la TSA no reconozca, ésta devolverá un mensaje de error de extensión no aceptada (unacceptedExtension). Más información en: RFC 2459

```
MessageImprint ::= SEQUENCE  {
     hashAlgorithm     AlgorithmIdentifier,
     hashedMessage     OCTET STRING
}
```

Campo	Descripción
hashAlgorithm	OID del algoritmo hash: El algoritmo de hash indicado en *hashAlgorithm* debería ser uno conocido por la TSA. También comprobará que sea suficientemente fuerte. Si la TSA no reconoce el algoritmo usado o piensa que es débil, entonces la TSA denegará el servicio al cliente devolviendo un **pkiStatusInfo** de 'bad_alg'.
hashedMessage	Este campo contiene el hash de los datos que se quiere sellar. La longitud del hash tiene que coincidir con la longitud de hash del algoritmo utilizado

El mensaje ***Timestamp Request*** no identifica al cliente, y esta información no es validada por la TSA. En el caso en que la TSA requiera su identidad deberá utilizar un mecanismo alternativo de identificación o autenticación.

3.2.1 Proceso de sellado

Durante el proceso de sellado, la TSA sistema realiza diferentes acciones, primero realiza una revisión de la petición, verificando la correcta estructuración del objeto TimeStampRequest y el origen de la misma. Durante esta verificación se comprueba que se han introducido los parámetros esperados como el algoritmo de hash (MD5, SHA-1, SHA-256 y RIPE160) y la política de sellado y que son correctos.

Posteriormente se obtiene de la fuente segura de tiempo y se genera el token de tiempo que es firmado electrónicamente por el proveedor de firma asociado al usuario ya sea dispositivo de seguridad hardware (HSM) o dispositivo de seguridad para certificados software. Actualmente el software de TSA sólo admite un único **Proveedor de firma** (en la tabla TSA_SIGNERS). Este proveedor de firma permite configurar en su campo SIGNERTYPE si queremos utilizar "Certificados de firma en soporte software" o con "Certificados de firma basado en modulo de seguridad HSM".

3.3 TIMESTAMP RESPONSE

Es la respuesta que la TSA da a una mensaje *time stamp request*. Tiene la siguiente representación:

```
TimeStampResp ::= SEQUENCE  {
     status            PKIStatusInfo,
     timeStampToken    TimeStampToken    OPTIONAL
}
```

Campo	Descripción
Status	Estado de la respuesta. Ver sección 3.2.3 del RFC 2510
timeStampToken	Este campo que contiene la marca de tiempo generado. Es una estructura **ContentInfo** que encapsula información firmada en una estructura **TSTInfo**. Está definida en la RFC 2630.

```
PKIStatusInfo ::= SEQUENCE {
     status            PKIStatus,
     statusString      PKIFreeText       OPTIONAL,
     failInfo          PKIFailureInfo    OPTIONAL
}
```

Campo	Descripción
Status	Estado de la respuesta: • **granted(0):** Marca de tiempo presente. • **grantedWithMods(1):** Marca de tiempo presente con modificaciones. • **rejection(2):** Petición rechazada • **waiting(3):** Esperando • **revocationWarning(4) :** Advertencia de revocación inminente • **revocationNotification (5):** Notificación de revocación
statusString	Puede usarse para indicar eventos de error
failInfo	Causas del fallo: • **badAlg(0):** Identificador de algoritmo no soportado • **badRequest(2):** Transacción no permitida o soportada • **badDataFormat(5):** Datos enviados con formato incorrecto • **timeNotAvailable(14):** Origen de tiempo no disponible • **unacceptedPolicy(15):** Política solicitada no soportada • **unacceptedExtension(16):** Extensión no soportada • **addInfoNotAvailable(17):** Información adicional no disponible • **systemFailure(25):** Error del sistema

```
TSTInfo ::= SEQUENCE  {
     version            INTEGER  { v1(1) },
     policy             TSAPolicyId,
     messageImprint     MessageImprint,
     serialNumber       INTEGER,
     genTime            GeneralizedTime,
     accuracy           Accuracy                OPTIONAL,
     ordering           BOOLEAN              DEFAULT FALSE,
     nonce              INTEGER                   OPTIONAL,
     tsa                [0] GeneralName           OPTIONAL,
     extensions         [1] IMPLICIT Extensions    OPTIONAL
```

```
}
```

Campo	Descripción
Version	Versión de la respuesta TimeStamp (v1)
ReqPolicy	OID de la política de la TSA Indica la política de la TSA bajo la cual se proporciona el sello, si se ha generado el sello, será igual al del mensaje de petición
messageImprint	OID del algoritmo hash y el valor del hash de los datos. Debe tener el mismo valor que el campo correspondiente de la petición.
SerialNumber	Es un entero asignado por la TSA y debe ser único para cada sello que genere. Por tanto, un sello será identificado por el nombre de la TSA que lo generó y el número de serie asignado. Permite hasta 160 bits
genTime	Es el instante de tiempo en el que se creó el sello. Tanto ISO como el IETF expresan el instante de tiempo referido a la escala *UTC*, para evitar confusiones con las horas locales. El formato debe ser el siguiente: CC YY MM DD hh mm ss Z • CC representa el siglo (19-99) • YY representa el año (00-99) • MM representa el mes (01-12) • DD representa el día (01-31) • hh representa la hora (00-23) • mm representa los minutos (00-59) • ss representa los segundos (00-59) • Z viene de *zulu*, que es como se conoce a la escala *UTC*
accuracy	Representa la desviación del tiempo UTC contenido en genTime, en los casos que sea necesario, proporciona una precisión incluso de microsegundos: `Accuracy ::= SEQUENCE { seconds [1] Integer OPTIONAL, millis [2] Integer (1..999) OPTIONAL, micros [3] Integer (1..999) OPTIONAL, }` Cuando este campo no está presente la precisión puede obtener a través de otros métodos, por ejemplo **TSAPolicyId.**
ordering	Si falta el campo ordering o está presente y tiene valor false, entonces el campo **genTime** solo indica el momento en el que la marca de tiempo ha sido creada por la TSA. En este caso, el orden de la marcas de tiempo emitidas por una misma TSA o distintas TSAs solo es posible cuando la diferencia entre el genTime de la primera marca de tiempo es mayor que la suma de las precisiones del genTime de cada marca de tiempo.
nonce	El **nonce** es un número aleatorio con una elevada probabilidad de que el cliente lo genere una única vez (entero de 64 bits). Debe tener el mismo valor que el campo correspondiente de la petición.
Tsa	Identificador de la TSA
extensions	Es una forma de permitir añadir nuevos campos en el futuro. Más información en: RFC 2459

3.4 VALIDATE REQUEST

Es el mensaje que una entidad envía a una TSA cuando quiere comprobar la validez y la autenticidad de un sello. En la RFC 3161 del IETF no se contempla el proceso de verificación del sello de tiempo:

```
ValidateRequest ::= SEQUENCE {
     version     INTEGER { v1(0) },
     tst         TimeStampToken,
     requestID   [0] OCTET STRING OPTIONAL
```

```
}
```

- - *tst*: contiene el sello que se quiere verificar
- - *requestID*: identificador que se utiliza para vincular una petición con su respuesta

3.4.1 Validate Reply

La TSA envía este mensaje como respuesta a una petición de verificación:

```
ValidateReply ::= SEQUENCE {
      version      INTEGER { v1(0) },
      status       PKIStatusInfo,
      tst          TimeStampToken,
      requestID    [0] OCTET STRING OPTIONAL
}
```

3.5 EJEMPLO DE CONEXIÓN EN JAVA

A continuación se describe un ejemplo de conexión a la TSA de EDICOM mediante código JAVA y utilizando las librerías *bouncycastle* (http://www.bouncycastle.org/latest_releases.html): bcprov-jdkxx-145.jar y bctspjdkxx-145.jar

```
package com.edicom.map.net;
import java.io.*;
import java.math.*;
import java.net.*;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;

import org.bouncycastle.asn1.cmp.*;
import org.bouncycastle.asn1.x509.*;
import org.bouncycastle.tsp.*;
import org.bouncycastle.util.encoders.Base64;

public class TimestampClient {
      /** URL of the Time Stamp Authority */
      protected String tsaURL;
      /** TSA Username */
      protected String tsaUsername;
      /** TSA password */
      protected String tsaPassword;
      /** Estimate of the received time stamp token */
      protected int tokSzEstimate;
      /**
      * Creates an instance of a TSAClient that will use BouncyCastle.
```

```
	* @param url String - Time Stamp Authority URL (i.e.
	"http://tsatest1.digistamp.com/TSA")
	*/

	public TimestampClient(String url) {
		this(url, null, null, 4096);
	}
	/**
	* Creates an instance of a TSAClient that will use BouncyCastle.
	* @param url String - Time Stamp Authority URL (i.e.
	"http://tsatest1.digistamp.com/TSA")
	* @param username String - user(account) name
	* @param password String - password
	*/

	public TimestampClient(String url, String username, String password) {
	this(url, username, password, 4096);
	}
	/**
	* Constructor.
	* Note the token size estimate is updated by each call, as the token
	* size is not likely to change (as long as we call the same TSA using
	* the same imprint length).
	* @param url String - Time Stamp Authority URL (i.e.
	"http://tsatest1.digistamp.com/TSA")
	* @param username String - user(account) name
	* @param password String - password
	* @param tokSzEstimate int - estimated size of received time stamp
	token (DER encoded)
	*/

	public TimestampClient(String url, String username, String password, int
tokSzEstimate) {
			this.tsaURL = url;
			this.tsaUsername = username;
			this.tsaPassword = password;
			this.tokSzEstimate = tokSzEstimate;
	}

	/**
	* Get timestamp token - Bouncy Castle request encoding / decoding layer
	*/

	protected byte[] getTimeStampToken(byte[] imprint) throws Exception {
		byte[] respBytes = null;
		try {
			// Setup the time stamp request
```

```
                    TimeStampRequestGenerator tsqGenerator = new
                    TimeStampRequestGenerator();
                    tsqGenerator.setCertReq(true);
                    // tsqGenerator.setReqPolicy("1.3.6.1.4.1.601.10.3.1");
                    BigInteger nonce =
BigInteger.valueOf(System.currentTimeMillis());
                    TimeStampRequest request =
tsqGenerator.generate(X509ObjectIdentifiers.id_SHA1.getId(),imprint,nonce);


                    byte[] requestBytes = request.getEncoded();


                    // Call the communications layer
                    respBytes = getTSAResponse(requestBytes);


                    // Handle the TSA response
                    TimeStampResponse response = new
TimeStampResponse(respBytes);


                    // validate communication level attributes (RFC 3161
PKIStatus)
                    response.validate(request);
                    PKIFailureInfo failure = response.getFailInfo();
                    int value = (failure == null) ? 0 : failure.intValue();
                    if (value != 0) {
                           // @todo: Translate value of 15 error codes defined
by PKIFailureInfo to string
                           throw new Exception("[invalid.tsa.1.response.code]
url: " + tsaURL + ". status: " + String.valueOf(value));
                    }
                    // @todo: validate the time stap certificate chain (if we
want
                    //     assure we do not sign using an invalid timestamp).
                    // extract just the time stamp token (removes communication
status info)
                    TimeStampToken tsToken = response.getTimeStampToken();
                    if (tsToken == null) {
                           throw new
Exception("[tsa.1.failed.to.return.time.stamp.token] url: " + tsaURL + ".
status: " + response.getStatusString());
                    }
                    TimeStampTokenInfo info = tsToken.getTimeStampInfo(); // to
view details


                    byte[] encoded = tsToken.getEncoded();
                    long stop = System.currentTimeMillis();


                    // Update our token size estimate for the next call (padded
to be safe)
                    this.tokSzEstimate = encoded.length + 32;
                    return encoded;
             } catch (Exception e) {
```

```
                        throw e;
                } catch (Throwable t) {
                        throw new Exception("[failed.to.get.tsa.response.from.1]
url: " + tsaURL + ". status: " + t);
                }
        }

        /**
        * Get timestamp token - communications layer
        * @return - byte[] - TSA response, raw bytes (RFC 3161 encoded)
        */

        protected byte[] getTSAResponse(byte[] requestBytes) throws Exception {
                // Setup the TSA connection
                URL url = new URL(tsaURL);
                URLConnection tsaConnection;
                tsaConnection = (URLConnection) url.openConnection();
                tsaConnection.setDoInput(true);
                tsaConnection.setDoOutput(true);
                tsaConnection.setUseCaches(false);
                tsaConnection.setRequestProperty("Content-Type",
"application/timestampquery");
        //tsaConnection.setRequestProperty("Content-Transfer-Encoding", "base64");
        tsaConnection.setRequestProperty("Content-Transfer-Encoding", "binary");
        if ((tsaUsername != null) && !tsaUsername.equals("") ) {
                String userPassword = tsaUsername + ":" + tsaPassword;
                tsaConnection.setRequestProperty("Authorization", "Basic " +
                        new String(Base64.encode(userPassword.getBytes())));
                }
                OutputStream out = tsaConnection.getOutputStream();
                out.write(requestBytes);
                out.close();

                // Get TSA response as a byte array
                InputStream inp = tsaConnection.getInputStream();
                ByteArrayOutputStream baos = new ByteArrayOutputStream();
                byte[] buffer = new byte[1024];
                int bytesRead = 0;
                while ((bytesRead = inp.read(buffer, 0, buffer.length)) >= 0) {
                        baos.write(buffer, 0, bytesRead);
                }
                byte[] respBytes = baos.toByteArray();

                String encoding = tsaConnection.getContentEncoding();
                if (encoding != null && encoding.equalsIgnoreCase("base64")) {
                        respBytes = Base64.decode(new String(respBytes));
                }
```

```
		return respBytes;
	}
	public byte[] generateImprint(byte[] data) throws NoSuchAlgorithmException
{
		MessageDigest md = MessageDigest.getInstance("SHA-1");
		return md.digest(data);
	}
	public byte[] generateImprintFromFile(String filename) throws Exception {
		byte[] data = loadBytesFromFile(filename);
		return generateImprint(data);
	}
	public byte[] loadBytesFromFile(String name) {
		FileInputStream in = null;
		try {
			in = new FileInputStream(name);
			ByteArrayOutputStream buffer = new ByteArrayOutputStream();
			int ch;
			while ((ch = in.read()) != -1)
				buffer.write(ch);
			return buffer.toByteArray();
			} catch (IOException e) {
				if (in != null) {
					try {
						in.close();
					} catch (IOException e2) {
					}
				}
			return null;
		}
	}

	public void writeFile(byte[] data, String filename) throws IOException{
		OutputStream out = new FileOutputStream(filename);
		try {
			out.write(data);
			} finally {
			out.close();
		}
	}
	public static void main(String[] args) {
		String url = "http://tsa.acedicom.edicomgroup.com:9026/";
		if (args.length > 0) {
			try {
				String user = "usuario";
				String password = "password";
				TimestampClient tsc = new TimestampClient(url, user,
password);
```

```
                        byte[] imprint =
tsc.generateImprintFromFile(args[0]);
                        byte[] response = tsc.getTimeStampToken(imprint);
                        tsc.writeFile(response, args[0] + ".tsr");
                    }
                    catch (Exception e) {
                        e.printStackTrace();
                    }
                }
        }
}
```