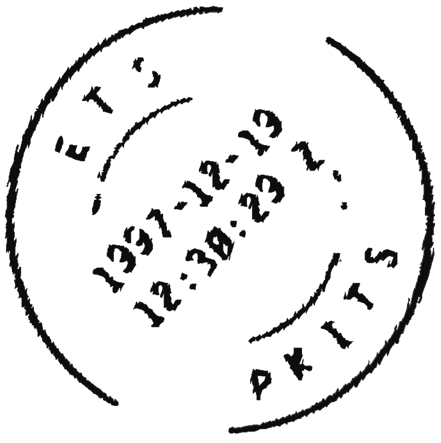


PKITS
Public Key Infrastructure with Time-Stamping Authority

ETS PROJECT: 23.192

Deliverable D7c
Description and Results of the Multimedia Time-Stamping Protocol Implementations

Produced by: ***FNMT***
Date of issue: 28th December ***1998***
Revision Number: ***10***

TABLE OF CONTENTS

GLOSSARY OF TERMS

Specifications:	
SHALL	Essential requirement. A requirement must be fulfilled or a feature implemented wherever this term occurs. The designer is requested, however, to indicate if one or more “shall requirements” would increase the cost or time unreasonably in relation to the total cost or design cost, in which case the specification may have to be revised.
SHOULD	Important requirement. Shall be implemented without or with minimum extra cost. Valid reasons in particular circumstances may allow ignoring such requirements.
MAY	Optional requirement. From case to case, it should be decided whether implementing it or not, in any case without exceeding the budget planned for the related activity.

Technical:	
API	Application Programming Interface
GIF	Graphical Image Format
IP	Internet protocol
JCE	Java Cryptography Extensions
JPEG	Joint Photographic Expert Group
NTP	Network Time Protocol
PIF	Portable Image Format
PKCS	Public Key Cryptography Standards
SHA	Standard Hash Algorithm
TCP	Transport Control Protocol
TSA	Time-Stamping Authority
TSS	Time-Stamping Service
TTP	Trusted Third Party

1. EXECUTIVE SUMMARY

This document describes the activities performed in order to validate and provide performance measurements of the time-stamping protocols and supporting algorithms developed within the PKITS project.

The testbed architecture provides a proof of concept, and a demonstration to start getting hands-on experience as a preliminary step towards interaction with time servers and use of available network time protocol tools.

The testbed for time-stamping multimedia information consists of a generic TSA that provides a basic time-stamping service, plus a specific TSA that provides an interface to deal with multimedia information, and a sample client application that uses the previous TSAs to provide a whole service. The testbed runs on Internet using TCP/IP sockets to connect the components. Graphical user interfaces for each component permit the parameterisation of the components, and the use of the testbad as a demonstration of the service.

NTP protocol is used over Internet to provide a time service synchronised with an external reliable source. Provision is made to track synchronisation, and stop TSA operation would the synchronisation fail over a specified accuracy.

The testbed has been used to acquire hands-on experience on the provision of a TSS, evaluating the complexity of the software, validating the protocols and formats proposed along the PKITS Project, and measuring the throughput that may be obtained using off-the-shelf components.

As a final benefit, a demonstration of the service is publicly available on Internet.

2. REFERENCES

[CRYPTIX]	Cryptix Strong Cryptography library in Java version 3.0.3, http://cryptix.moremagic.com/.
[JPEG]	ISO/IEC 10918 (JPEG), “Information Technology – Digital Compression and Coding of Continuous Tone Images”, July 1992.
[MINICODE]	http://www.omniplanar.com/minicode.html
[PKITS D3]	Architecture of Time-Stamping Service and Scenarios of Use: Service and Features, Deliverable D3 of ETS Project 23.192 Public Key Infrastructure with Time-Stamping Authority, May, 1998.
[PKITS D4]	Time-Stamping Service Functional Specification and Protocols for Unstructured Data, Deliverable D4 of ETS Project 23.192 Public Key Infrastructure with Time-Stamping Authority, June 1998.
[PKITS D6]	Time-Stamping Service Functional Specification and Protocols for Multimedia Information, Deliverable D3 of ETS Project 23.192 Public Key Infrastructure with Time-Stamping Authority, August 1998.

3. DESCRIPTION OF THE MULTIMEDIA TIME-STAMPING TESTBED

In the multimedia time-stamping testbed several components have been developed in order to verify that the distribution of services and tasks among the different subsystems that has been proposed in previous deliverables [PKITS D4, PKITS D6] is practical. As an additional benefit, we have gained experience in the implementation of these services and their corresponding user interface elements.

In this testbed we deal with static, colour JPEG images, although performance measurements have been accomplished in order to validate the proposed protocols and algorithms for the time-stamping of live (10 or more frames per second) video sequences.

The prototyped subsystems are a raw-data TSA (with no knowledge whatsoever of the document structure), a multimedia TSA (able to insert time-stamp tokens and parse multimedia information), and a multimedia time-stamping client, able to request and verify image time-stamp tokens.

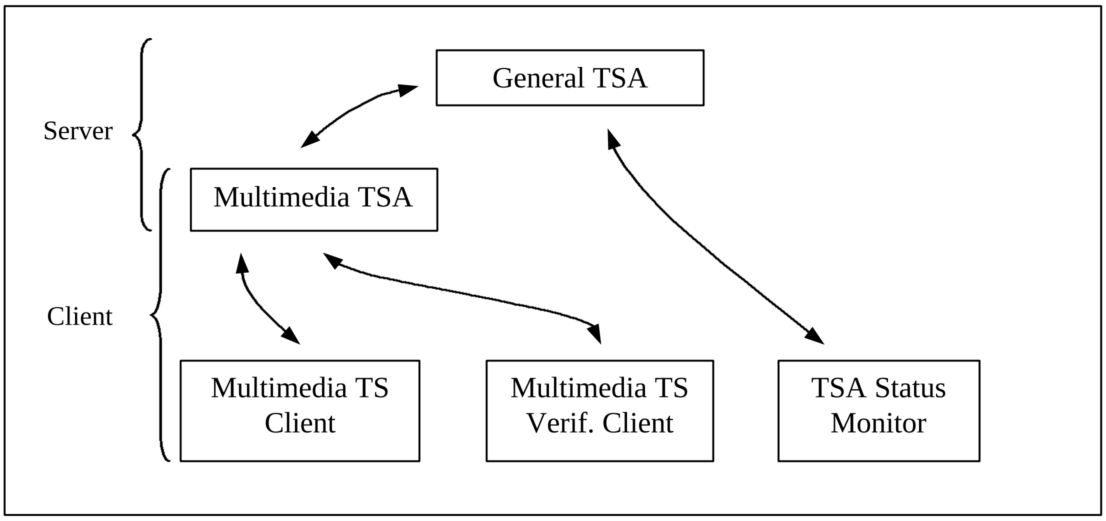
The system generates the necessary digital signatures using cryptographic libraries installed in the workstation. The workstation real-time clock is used as time source, with periodic synchronisation via the NTP protocol. The whole time-stamping system is periodically synchronised with other TSAs on a regular basis.

3.1. TESTBED ARCHITECTURE

Five subsystems have been implemented in the multimedia time-stamping testbed:

1. General TSA
2. Multimedia TS Server
3. Multimedia TS Request Client
4. Multimedia TS Verification Client
5. TSA Status Monitor

Depending on the usage scenario, two different architectures have been identified, whose single difference lies in the place where the Multimedia TS Server is located. We will study the differences among these architectures in a later section, after the description of the subsystems.



3.2. SERVER-SIDE COMPONENTS

The server-side time-stamping subsystems run inside a Java 2 (formerly known as “Java 1.2”) runtime environment. All the server-side components are applications, communicating with each other and with the clients via TCP/IP ports selected by the service administrator.

3.2.1. General TSA

The general TSA is a Java application that implements the centralised request, retrieval and verification protocols described in [PKITS D4], both in the simple and linking versions. Several concurrent accesses can be processed simultaneously (a TSA thread is awaked for each received request, with a maximum number of threads controlled by the TSA administrator) in all but the linking protocol time-stamp token request, where the nature of the linking prevents doing so.
The general TSA does not contain any multimedia-specific code, and it could be used to time-stamp any kind of information.
As subsystems of the general TSA we can enumerate the time source, the time synchronisation daemon and the cryptographic library; all of these subsystems are essential for the correct operation of the general TSA.

3.2.1.1. Time Source

The workstation real-time clock is used as time source. Its value is obtained via an ordinary system call of the Java Runtime Environment. The time synchronisation daemon (described below) ensures that the time deviation from the primary time source is always within 0.125 seconds; if synchronisation could not be performed due to unavailability or severe deterioration of the communications channel, the TSA would suspend its service until normal operation.

3.2.1.2. Time Synchronisation Daemon

Time synchronisation of the workstation real-time clock is performed via the NTPv3 protocol, which ensures that the local time deviation is minimal with respect to a main time source that is configurable by the user. In the testbed a time server was set-up on the local network.
Statistical analysis of time deviations has led us to conclude that the maximum drift of the used workstation real-time clock is 6 seconds per day, or 6s / 86 400s = $6.944 \cdot 10^{-5}$. Synchronisation every 30 minutes guarantees a maximum time deviation of 1800s · ($6.944 \cdot 10^{-5}$) = 0.125 s with respect to the primary time source.

3.2.1.3. Cryptographic Library

Since all the exchanged messages are PKCS#7 SignedData constructs, heavy use is made of digital signatures in the provision of the time-stamping services. Therefore, it seemed that a “pure Java” digital signature class package would severely reduce the performance of the TSA, both in terms of latency and of requests processing speed. Since Sun’s Java Cryptography Extensions (JCE) API cleanly separates the use of cryptographic primitives and the provision of those services, the TSA can work in conjunction to any JCE service provider. The multimedia time-stamping testbed uses Cryptix [CRYPTIX] JCE native service provider for the Intel processors.

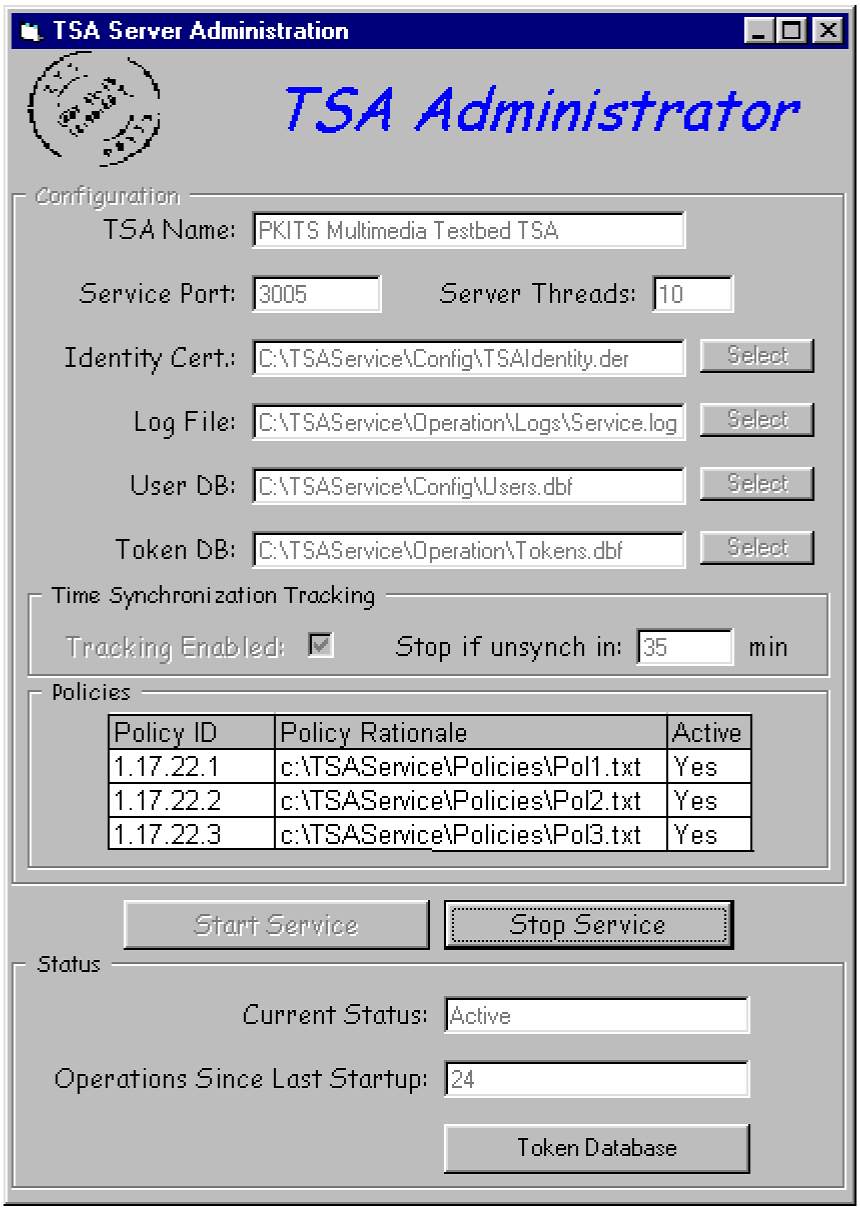
3.2.1.4. User and Time-Stamp token databases

Installed in the General TSA there is a user database against which to check time-stamp requests. In the testbed this database has been implemented as a series of files with the user identities, all together in a directory that can be chosen by the TSA administrator.

There is also a time-stamp token database where the TSA stores all the produced time-stamp tokens. In the testbed this token database is also implemented as a series of files, one for each token, all located in a directory selected by the system administrator.

3.2.1.5. Graphical User Interface

The following figure presents the graphical user interface of the General TSA.



Configuration

TSA Name:	PKITS Multimedia Testbed TSA
Service Port:	3005
Server Threads:	10
Identity Cert.:	C:\TSAService\Config\TSAIdentity.der
Log File:	C:\TSAService\Operation\Logs\Service.log
User DB:	C:\TSAService\Config\Users.dbf
Token DB:	C:\TSAService\Operation\Tokens.dbf

Time Synchronization Tracking

Tracking Enabled: ✓ Stop if unsynch in: 35 min

Policies

Policy ID	Policy Rationale	Active
1.17.22.1	c:\TSAService\Policies\Pol1.txt	Yes
1.17.22.2	c:\TSAService\Policies\Pol2.txt	Yes
1.17.22.3	c:\TSAService\Policies\Pol3.txt	Yes

Status

Current Status: Active

Operations Since Last Startup: 24

In the above screen some user interface elements are disabled, since the TSA was providing service and those parameters could not be modified with the TSA in Active status.

Although the interface attempts to be self-explanatory, notice the following facts:

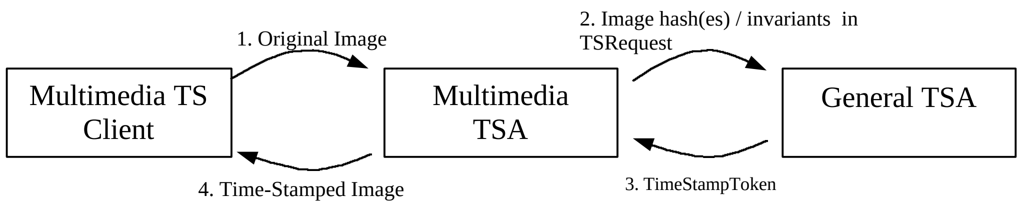
- It is possible to configure the service port used to receive time-stamp request from the client (in our setting, from the Multimedia TSA).

- The server has a pool of threads that service client requests as they arrive to the service port. If the underlying time-stamping process is inherently "non-concurrent," such as for example in the case of a linking protocol, the subsystem implementing that protocol must provide access synchronisation.

- There is a "time synch tracking" mechanism implemented in order to check that time synchronisation is being performed via NTP with a primary time server (the configuration of the NTP client is not part of the General TSA). The General TSA checks the success of the NTP synch operation, and if unable to properly synchronise the time in the selected period of time, it stops providing its service.

3.3. MULTIMEDIA TSA

The multimedia TSA can be run in the server machine as a Java application, in which case it acts as a bridge between the client of the image time-stamping protocol and the general TSA. It encapsulates all the multimedia-dependent operations, such as parsing of the incoming images and the insertion of time-stamp tokens in application header tags.



The operation of the Multimedia TSA is then the following:

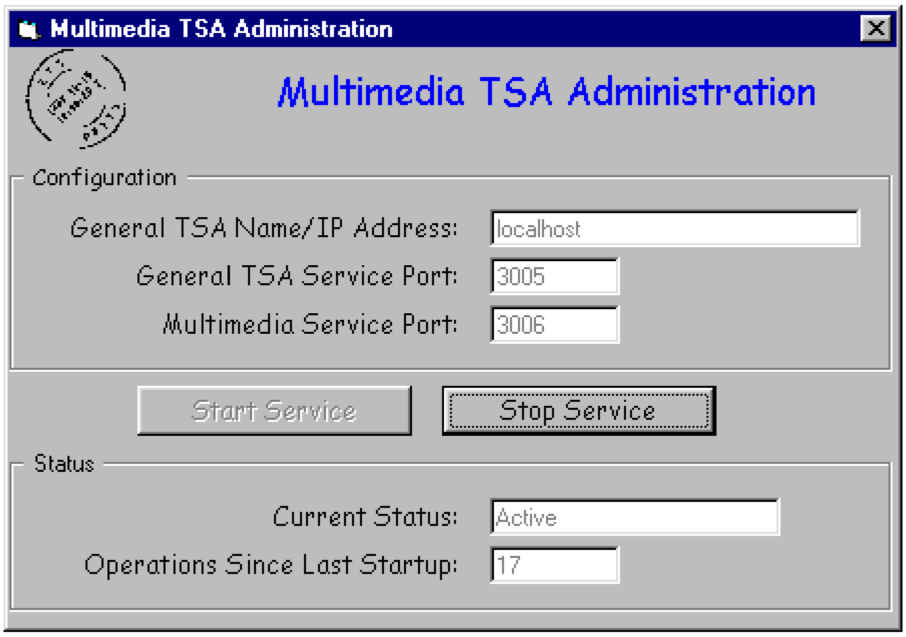
1. Computation of image hashes is performed with the message digest services offered by Sun's Java Cryptography Extensions API. We are using the Cryptix native service provider to compute the MD5 and SHA-1 hash algorithms. These hash values are inserted in a TimeStampRequest [PKITS-D4] and then encapsulated in a PKCS#7 SignedData construct that is sent to the General TSA.

2. The TSA then will return the corresponding TimeStampToken, encapsulated in a PKCS#7 SignedData construct.

3. The Multimedia TSA will insert the TimeStampToken SignedData in the first free application tag of the image.

4. The resulting time-stamped image is then returned to the Multimedia TS Client.

The following figure presents the graphical user interface for the Multimedia TSA:



Configuration

General TSA Name/IP Address: localhost

General TSA Service Port: 3005

Multimedia Service Port: 3006

Status

Current Status: Active

Operations Since Last Startup: 17

The Multimedia TSA communicates with its clients with the following protocol:

1. The client opens a TCP socket using the Multimedia TSA service port.
2. The client sends a protocol version identifier to the server. In this implementation, the ASCII string “PKITS-ITP0” is used as version identifier.
3. The client sends to the server a four-octet identifier of the image format. In our case, where the JPEG format is used, the ASCII identifier “JPEG” is used.
4. The client sends a four-octet value encoding the length of the image that will be transmitted. This number is encoded as an unsigned value, with the most significant octet first.
5. The server returns the protocol identifier, that must be the same received from the client (in our implementation, “PKITS-ITP0”).
6. The server returns a four-octet identifier of the returned image format. In our case, the return documents are also in the JPEG format, and therefore the identifier “JPEG” is used.
7. The server returns a four-octet value encoding the length of the image that will be transmitted. This number is encoded as an unsigned value, with the most significant octet first.
8. The server returns the time-stamped document.

3.4. CLIENT-SIDE COMPONENTS

The client-side time-stamping subsystems run inside a Java 2 (formerly known as “Java 1.2”) runtime environment. All the client-side components can be run as applications and as applets within a web browser. Nothing prevents several of the components appear in the same web page, so that the end-user can access all the functionality implemented in the testbed from a single page.
As mentioned previously, the Multimedia TSA can be executed in the server side or in the client side. The decision where to locate this component should be adopted taking into account performance considerations (is the client workstation powerful enough to perform the necessary computations?), communication time and cost (is it practical to send the original and time-stamped images across the network?) and confidentiality of the time-stamped multimedia information. Since operation of the Multimedia TSA when executing on the client is the same, we will concentrate on the remaining client-side components.

3.4.1. Multimedia TS Client

The Multimedia TS Client is a combined Java applet/application that is able to request the time-stamping of images to the Multimedia TSA. The steps performed by this component are the following:

1. The user selects the image to be time-stamped from the local file system.
2. The image is sent to the Multimedia TSA.
3. The time-stamped image returned from the Multimedia TSA is stored in the local file system where desired by the user.

In order to work, it is necessary to properly configure the Multimedia TS Client. Important parameters are the user's X.509 certificate, the Multimedia TSA host name (or IP address), and its service port. The following figure displays the start screen of the Multimedia TS Client.

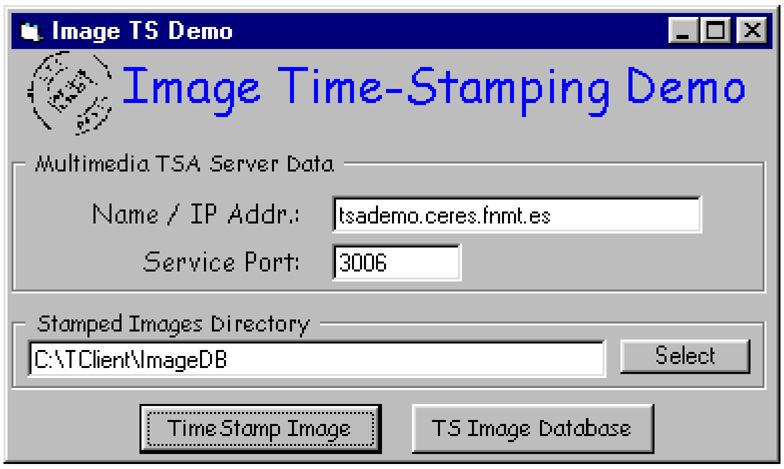


Image TS Demo

Image Time-Stamping Demo

Multimedia TSA Server Data

Name / IP Addr.: tsademo.ceres.fnmt.es

Service Port: 3006

Stamped Images Directory

C:\TClient\ImageDB

When the user presses the "Time Stamp Image" command button, a file chooser appears that lets him/her select the JPEG image to time stamp. If the operation succeeds, another file chooser will appear so that the user stores the time-stamped image in a convenient place. Another copy of the time-stamped image will be stored in the "Stamped Images Directory."

3.4.2. Multimedia TS Verification Client

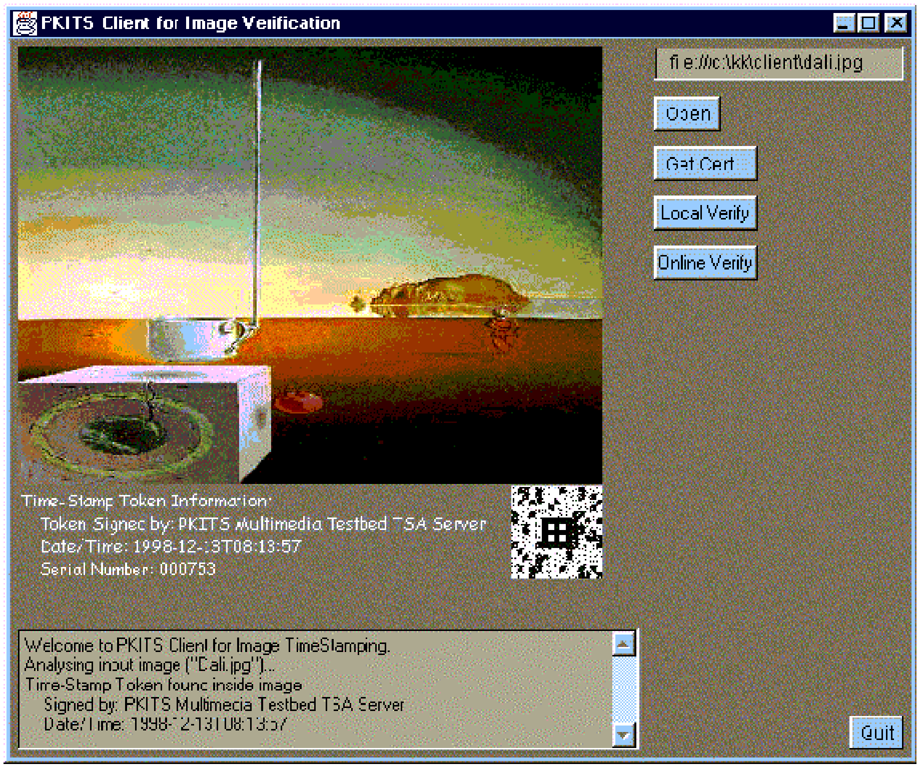
The Multimedia TS Verification Client is a combined Java applet/application that is able to perform both local and remote verification of time-stamp tokens inserted in images. This component is able to perform the following actions:

- Local verification of a time-stamped image.
  The digital signature present in the time-stamp token is verified, without connection to the General or Multimedia TSA. The Cryptix native service provider of the Java Cryptography Extension (JCE) is used, although any JCE-compatible package could be used.

- Online verification of a time-stamped image.
  This time a full verification of the time-stamped image is performed, according to the verification protocol specified in [PKITS-D4].

This component is a crucial one in the multimedia time-stamping testbed. Time-stamped images can be embedded in web pages with the help of this component, and the time-stamping information can transparently be presented to the user, also offering the possibility of local and online verification of the embedded time-stamp.

Important configuration parameters of the Multimedia TS Verification client are the TSA's X.509 certificate (local verification), the Multimedia TSA host name (or IP address), and its service port. The following figure displays the user interface of the Multimedia TS Verification Client.


PKITS Client for Image Verification
file://c:\kk\client\dali.jpg
Open
Get Cert
Local Verify
Online Verify
Quit

```
Time-Stamp Token Information:
   Token Signed by: PKITS Multimedia Testbed TSA Server
   Date/Time: 1998-12-13T08:13:57
   Serial Number: 000753
```

```
Welcome to PKITS Client for Image TimeStamping.
Analysing input image ("Dali.jpg")...
Time-Stamp Token found inside image
   Signed by: PKITS Multimedia Testbed TSA Server
   Date/Time: 1998-12-13T08:13:57
```

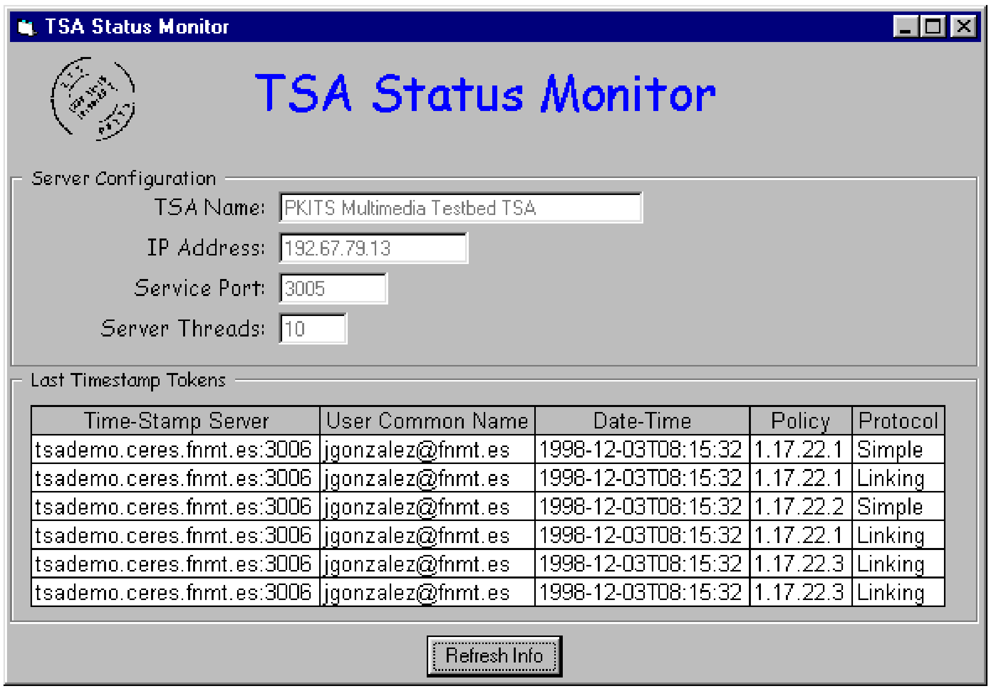
Observe that time-stamp token information is displayed under the image in machine-readable form, in a format known as MiniCode [MINICODE]. Other more sophisticated, higher-capacity 2D-barcodes could be used for the same purpose, but MiniCode has proved enough for the intended purposes. This 2-D code permits to print the photograph on paper, and still recover the time certificate without human re-keying.

3.4.3. TSA Status Monitor

In order to enable the remote monitoring of the TSA operation (not its administration), a status monitor has been implemented so that users can inspect the TSA operation in real-time. This TSA Status Monitor shows:

- The available policies and the corresponding time-stamp practice statements.
- The time-stamp tokens produced under each of the operation policies.
- The links stored in the TSA (for those policies that make use of them).
- The TSA operation parameters (IP address, service port, etc).

The following figure shows the user interface of the TSA Status Monitor:



Server Configuration

TSA Name:	PKITS Multimedia Testbed TSA
IP Address:	192.67.79.13
Service Port:	3005
Server Threads:	10

Last Timestamp Tokens

Time-Stamp Server	User Common Name	Date-Time	Policy	Protocol
tsademo.ceres.fnmt.es:3006	jgonzalez@fnmt.es	1998-12-03T08:15:32	1.17.22.1	Simple
tsademo.ceres.fnmt.es:3006	jgonzalez@fnmt.es	1998-12-03T08:15:32	1.17.22.1	Linking
tsademo.ceres.fnmt.es:3006	jgonzalez@fnmt.es	1998-12-03T08:15:32	1.17.22.2	Simple
tsademo.ceres.fnmt.es:3006	jgonzalez@fnmt.es	1998-12-03T08:15:32	1.17.22.1	Linking
tsademo.ceres.fnmt.es:3006	jgonzalez@fnmt.es	1998-12-03T08:15:32	1.17.22.3	Linking
tsademo.ceres.fnmt.es:3006	jgonzalez@fnmt.es	1998-12-03T08:15:32	1.17.22.3	Linking

Information on the current configuration is presented to the user, as well as information on the latest time-stamp tokens produced. Of course, this user interface does not permit the alteration of the administrative configuration of the TSA.

3.4.4. Web Service

The user side is provided as well as applet components that can be demonstrated on Internet. The service is publicly available on the PKITS web pages, and permits the same operations as previously described:

- Users may contribute JPEG images for time-stamping
- Users receive a time-stamped image, with the information embedded into JPEG tags, and explicit time certificate as a footer, and as a 2-D bar code
- Users may retrieve the TSA certificate
- Users may verify the time certificate locally
- Users may verify the time certificate on-line against the TSA
- TSA status may be accessed on-line

4. CONCLUSIONS

The testbed has permited to check the feasibility of the protocols and formats proposed in [PKITS D3, D4, and D6]. Only minor errors of detail have been identified in [PKITS D4], and will fire the issue of a new release of that deliverable with those details fixed. No modification has been required on [PKITS D3] or [PKITS D6].

Use of NTP to provide a reliable source of time for the local clock has proved to be easy to install and operate, and to provide a time accuracy quite adequate for the intended use of the testbed TSA.

Hands-on experience has permitted to derive some conclusions on understanding the architecture of a real TSS, as respects to multimedia information, to measure system performance, and to plan some further experiements.

4.1. IDENTIFIED ARCHITECTURES

Depending on the usage scenario, we have identified two different architectural configurations on the previously described subsystems. The difference lies in the side where the Multimedia TSA operates. The decision of which architecture to use depends on the performance of the server- and client-side machines, the confidentiality of the time-stamped information, and the bandwidth of the communications line between client and server.

4.1.1. Multimedia TSA Located On Client-Side

When the time-stamping process is not tolerant [PKITS-D6] and the complexity of the operations to be performed on the multimedia information is low, it seems beneficial to locate the Multimedia TSA in the client side. In this configuration, the communications between the client and server would only consist in small time-stamp requests and the response time-stamp tokens, whose size is much smaller than that of the multimedia information that is being time-stamped. If communication cost were also an issue, this architecture would result in an important performance gain.
This configuration seems profitable in the case when the multimedia information is confidential, since no use of cryptographic primitives seems practical in the alternative architecture if the server is to manipulate the received information in order to insert the time-stamp within the received data. Of course, this possibility deserves further study.

4.1.2. Multimedia TSA Located On Server-Side

When the time-stamping process being performed is tolerant [PKITS-D6], it is very likely that the complexity of the operations to be performed on the multimedia information is very high, making it necessary to locate the Multimedia TSA in the server side. Under this configuration, the communications between the client and server would consist of the multimedia information to be time-stamped, and as response the time-stamped information would be obtained. If communication cost were an issue, this architecture would degrade the system performance due to the high amount of exchanged information.

4.1.3. Separate Multimedia TSA

Lastly, the Multimedia TSA might be located on an independent server, providing a mixture of the benefits mentioned above. This 3-tier architecture may apply when a large collection of multimedia clients, with limited processing power, are running on a local network using a centralized multimedia server that communicates with a remote TSA for definite time-stamping.

4.2. PERFORMANCE

The testbed permits to obtain some performance figures that may be used to design a time-stamping service for the real world.

4.2.1. Used Equipment

The performance measurements presented in this section were obtained with the modules running on Intel Pentium 233 MHz computers with 64 Mbytes, running Windows 95. Communication took place through a 33 600 Kbits/second modem in the internal phone network.

4.2.2. Identity Certificates

The clients and the General TSA server have X.509v3 identity certificates issued by the TSA (the General TSA certificate is self-signed).

4.2.3. Performance of the full time-stamping operation

The full time-stamping process on a JPEG document is the concatenation of:

1. Verification of the user identity and time-stamp request
2. Hash computation on the original image
3. Time grabbing
4. Time-stamp token digital signature

About 10 milliseconds are needed to obtain a user identity certificate from the local user database. A binary search in the user database is performed in the testbed implementation, and therefore no significant increase is expected if the number of users grew to the order of thousands or even millions. Verification of the digital signature on the incoming time-stamp request PKCS#7 SignedData construct takes 3 milliseconds.

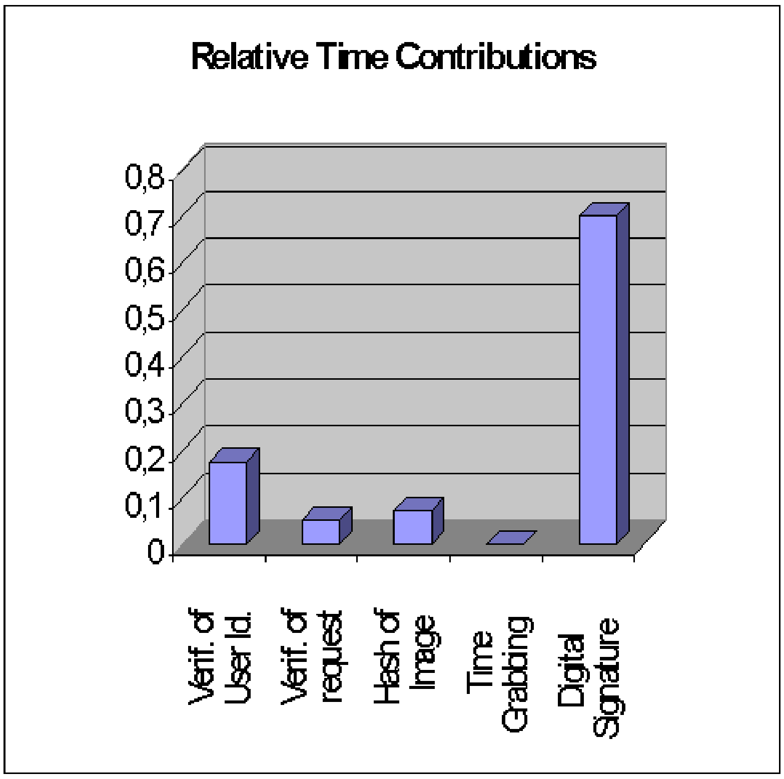
The computation of the hash of the incoming image depends on the selected hash algorithm. For a 16 kByte image, the hash computation times range from 2 milliseconds for MD5 to 6 milliseconds for SHA-1.

In the testbed implementation the time source is the PC real-time clock, synchronised with and NTP daemon with a remote time server. Grabbing of the time value is, therefore, an almost-instantaneous operation, with negligible cost.

The time needed to perform the digital signature operation on the time-stamp token in order to produce the response PKCS#7 SignedData construct depends on the selected algorithm. We have measured performance for RSA-512, RSA-1024 and DSA-1024. The times are (including SHA-1 hash computations) 12 milliseconds for RSA-512, 74 milliseconds for RSA-1024, and 41 milliseconds for DSA-1024.

Adding the partial times we see that generation of a single time-stamp token takes approximately 55 milliseconds in our testbed implementation, where the single optimisation is the use of a native Java cryptographic library [CRYPTIX]. Additional optimisations could include caching of client identity certificates, use of advanced pre-computations in the digital signature operations and, for the case where a huge number of certificates were issued per unit time, caching of the real-time clock value.

The following graph depicts the relative contribution of each operation to the full time-stamping process. (Average times of the operations have been used to produce the graph.)



4.3. Further Experiments

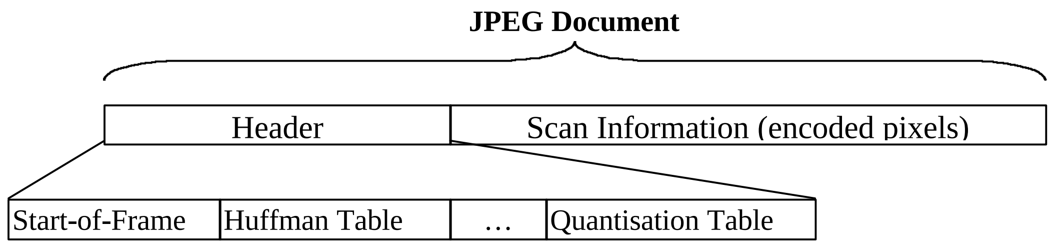
In the near future we plan to extend our experiences. Using the core testbed described above, and taking into account the performance figures reported, the following experiements will be carried along:

- other formats for still images, namely GIF, and PIF
- time-stamping of multimedia streams, such as audio, and real-time video (using one-time digital signatures as described in [PKITS D6]
- use of tolerant hashes, information that captures the core of the information, and survives small modifications and transformations

5. APPENDIX A: INSERTION OF THE TIME-STAMP TOKEN IN THE JPEG IMAGE

The main role of the Multimedia TSA is the insertion of the time-stamp certificate within the JPEG image in a transparent way, i.e., so that the resulting document still conforms to the JPEG format and is readable by JPEG-compliant software modules. In this section we explain how this insertion is performed.

According to the JPEG standard specification **[JPEG]**, a JPEG document consists of two clearly defined parts: a header and scan information. The header contains information such as image dimensions, bit-depth, and tables that are used in the quantisation and entropy-coding phases of the JPEG algorithms.

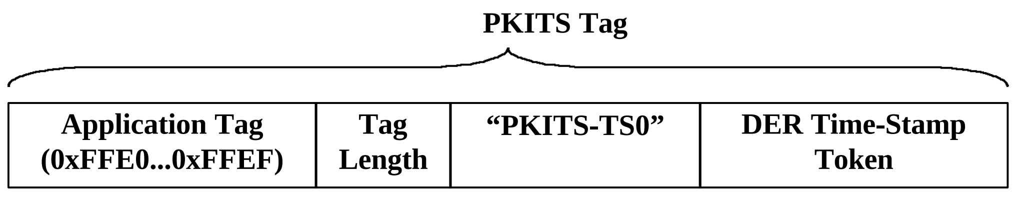


The header is formed by markers, or *tags*. The tags are buffers that begin with a header identifier which specifies the kind of information stored in the tag. Each tag identifier takes the form of two octets: the first octet is 0xFF, and the second octet is not equal to 0xFF or to 0x00. The quantisation table tag identifier, for example, is 0xFFDB.

Those tags whose length is variable indicate their actual length in a two-byte space that follows the header information; the length is stored in two octets, most-significant octet first. The length includes the two length-indication octets, but does not count the initial tag identifier. The maximum amount of information that can be stored in a JPEG marker is, therefore, 65 534 bytes. Of course, in practice tag lengths are always in the order of tens or hundreds of bytes.

An application data marker, or application tag, is included in the JPEG format specification to allow application-specific information to be inserted in the encoded image document. There are 16 application tag identifiers, taking values 0xFFE0 to 0xFFEF. In the multimedia time-stamping testbed there is no fixed application tag where the information is stored. When a JPEG image is to be time-stamped, the first unused application header is used, so that time-stamp renewal can be easily performed by application of the time-stamping protocol to an already time-stamped document.

Once the JPEG parser has localised the first unused application tag, it inserts the "PKITS-TS0" ASCII buffer, followed by the binary DER-encoded PKCS#7 SignedData time-stamp certificate.



6. APPENDIX B: OBJECT IDENTIFIERS USED IN THE MULTIMEDIA TESTBED IMPLEMENTATION

The object identifiers (OIDs) that were defined as "OBJECT IDENTIFIER { to be supplied }" in [PKITS D4] have been given the following values in the multimedia time-stamping testbed:

OID Name	OID Value
TimeStampRequestId	{ pkcs-7 2003 }
TimeStampTokenId	{ pkcs-7 2004 }
TimeStampId	{ pkcs-9 30 }
TimeStampProtocolId	{ pkcs-9 32 }
TimeStampValidityId	{ pkcs-9 34 }
TimeStampRequesterId	{pkcs-9 33 }
TimeStampSNId	{ pkcs-9 35 }
TimeStampLinksId	{ pkcs-9 38 }
TimeStampRenewalId	{ pkcs-9 40 }

pkcs-7 = {iso (1) member-body (2) US (840) rsadsi (113549) pkcs (1) 7 }
pkcs-9 = {iso (1) member-body (2) US (840) rsadsi (113549) pkcs (1) 9 }