Trusted Time™ API Software Development Kit

USER GUIDE, API VERSION 1.0

Revision Number
June, 2000

Document Part Number:

datum eBusiness Solutions

CONFIDENTIAL AND PROPRIETARY

Legal Notices

This product is classified by the U.S. Department of Commerce as Retail Product Encryption Software and is eligible for license exception ENC under sections 740.17 (A)(3) and (A)(4) of the Export Administration Regulations.

Contractor/manufacturer:

Datum eBusiness Solutions

10 Maguire Road, S-120, Lexington, MA 02421-3110 USA

+1(781)372-3600

www.datum.com

Trusted Time™ API Software Development Kit

Trusted Time™ API: Introduction

API: An Introduction

This document provides a detailed description of the **Datum EBS Trusted Time™ Application Programming Interface (TTAPI) Software Development Kit (SDK)**. The SDK provides an interface for software developers to the Datum Trusted Time Infrastructure (TTI), through the Trusted Time StampServer (TSS).

This document is aimed at describing the Trusted Time API and is not intended to give readers a comprehensive view of the Datum TTI architecture. It is recommended that readers become familiar with the TTI before reading this document. For a high-level definition of the TTI, including functions, architecture, main elements, and protocols, please refer to *Appendix B*.

Readers and software developers who use this SDK should have a fundamental understanding of public-key cryptography. Please refer to *Appendices A* and *B* of this document for more information.

This document provides:

- Instructions for installing the SDK
- Detailed information of TTAPI
- Guidelines for TTAPI application development
- Sample code

Glossary of Acronyms

These are the acronyms used throughout this document. For ease of use, they are also listed in *Appendix C*.

Table: Trusted Time™ Acronyms

Acronym	Definition
APC	Application Certificate
API	Application Program Interface
CA	Certification Authority or Certificate Authority
CRL	Certificate Revocation List
CRM	Certificate Request Message
DFC	Datum Factory Certificate
DFCA	Datum Factory Certificate Authority
DSA	Digital Signature Algorithm specified in DSS
DSS	Digital Signature Standard (FIPS 186)
DS/NTP	Datum Secure Network Time Protocol
DTT	Datum Temporal Token
ENMTMS	Element Manager
FIPS	Federal Information Processing Standard
GPS	Global Positioning System
IETF	Internet Engineering Task Force
NIST	National Institute of Standards and Technology
NMI	National Measurement Institute(s)
NMIServer	National Measurement Institute Server
NOC	Network Operations Center
NTMS	Network Time Management System
NTP	Network Time Protocol (RFC 1305)
OCSP	Online Certificate Status Protocol
OCSPROCSP	Responder
OEM	Original Equipment Manufacturer
OID	Object Identifier
PKI	Public Key Infrastructure
PLB	Private Label Branch
PSTN	Public Switched Telephone Network
RA	Registration Authority
SNMP	Simple Network Management Protocol
SSL	Secure Sockets Layer
TCCert	Time Calibration Certificate
TLS	Transport Layer Security
TMC	Trusted Master Clock
TPC	Third Party Certificate
TPCA	Third Party Certification/Certificate Authority
TSA	Time Stamp Authority
TSR	Time Stamp Request
TSS	TT StampServer
TST	Time Stamp Token
TT	Trusted Time
TTAPI	Trusted Time API
TTDS	Trusted Time Distribution Service
TTI	Trusted Time Infrastructure
TTP	Trusted Time Product
UDP/IP	User Datagram Protocol/Internet Protocol
UTC	Coordinated Universal Time

Chapter 1: Product Overview

Product Overview

The TTAPI is a set of functions that enable the user to develop TTAPI applications. These applications have the primary function of allowing the user to communicate with the TT StampServer, which is a PCI module that drops into a Windows machine. The TTAPI is used by software developers to enable their applications to obtain and use Time Stamp Tokens from a TSS-equipped server or Time Stamp Authority. Applications using the TTAPI range from Time Stamp Authorities to general e-commerce applications. Some applications use all TTAPI capabilities while others use only a subset.

Functions provided by the TTAPI include:

- TSS configuration
- Cryptographic parameter and key generation
- Key backup
- Creation and export of Temporal Tokens
- Creation and export of Time Stamp Tokens
- And TSS test

The TTAPI supports the cryptographic mechanisms that create hashes, validate token and certificate signatures, and establish encrypted and authenticated sessions.

The TTAPI function set uses the Datum Secure Network Time Protocol (DS/NTP) to speak to the TMC. Please refer to *Appendix A*.

The TTAPI function set is made to be used with a streamlined version of Windows Operating System and the Solaris Operating System. The TTAPI set has the same syntax for both operating systems and is installed differently for each one.

The TSS is available with a variety of software suitable for user with either Microsoft Windows NT or SUN OS. The kit includes drivers for low-level access as well as software programs for configuring and accessing the TSS Card.

Trusted Time Infrastructure Overview

The ability to time stamp e-documents is a fundamental business requirement. Critical to timestamping is the level of trust required in the actual source of time itself:

- How can a business be certain that the time contained in a time stamp is accurate?
- If multiple partners are conducting business transactions requiring time stamps, whose time will they agree to use?
- What proof exists that a server's built-in local clock has not been adjusted to an arbitrary value?
- Has a commonly-used time reference such as GPS been spoofed to indicate a different time?

Most time stamp solutions today ignore both the issue of time source trust and the universal nature of time. So inconsistencies proliferate throughout even the largest and most global business transactions.

To solve these problems, Datum eBusiness Solutions has developed a range of Trusted Time™ Products capable of securely and verifiably distributing time from a country's legal time source down to the local time stamp applications themselves. Datum Trusted Time is time that is certified and traceable to a specific legal time source—a country's National Measurement Institute (NMI). (In the United States, the National Institute of Standards and Technology (NIST) is legally established as the NMI.) Therefore, time distributed by Datum's Trusted Time Products cannot be spoofed or altered.

The following sections provide a detailed technical description of the Datum Trusted Time Products and how they are used to create a Trusted Time Infrastructure to securely distribute time and issue IETF-compliant time stamps.

See *Appendix B* for more information about TTI.

Universal Coordinated Time and Its Distribution

Universal Coordinated Time (UTC) is the world standard for time. The origin of UTC is the International Timing Authority (ITA) known as the International Bureau of Weights and Measures (BIPM) in France. National Timing Authorities in various countries discipline their atomic clocks to be within a few nanoseconds of that of BIPM's UTC. Each NMI then acts as a source of UTC for its country. By international agreement, the NMIs maintain audit records of their synchronization with BIPM UTC, thus providing verifiable sources of UTC within their countries. These NMIs supply time to hierarchies, or strata, of lower clocks that ultimately make UTC available to applications. This timing distribution hierarchy is summarized in Figure 1. While the ITA–NMI timing relationship is documented and audited, the timing distribution hierarchy from the NMI down to the application has traditionally not been secure.

TTI Functionality

The TTI is designed to provide the following functionality:

- Cryptographic signing of the National Measurement Institute (NMI) time to create trusted time.
- Reliable, trusted and traceable distribution of the trusted time from the NMI to local host-based applications.

- Optional time stamping of data provided to the local TTI element by the local application in a secure and reliable manner using the trusted time.

NOTE: The TTI-distributed trusted time is directly traceable back to the NMI for accuracy and non-repudiation purposes. For the purpose of this document, the phrase “trusted time” refers to trusted UTC time although the TTI could distribute other versions of time as well.

TTI Trust Model

To protect and authenticate UTC distribution, Datum’s Trusted Time Products (TTP) overlay onto the timing hierarchy, shown in Figure 1, to create a Trusted Time Infrastructure (TTI) that provides the necessary security and audit mechanisms at each stratum level to enable the trusted distribution of UTC from the NMI to the application. The Datum products comprising the TTI consist of: the NMI Trusted Time Server (NMIServer), the Trusted Master Clock (TMC), the TT StampServer (TSS), and the Network Time Management System (NTMS). The NMIServer, TMC and TSS are used to securely distribute time from the NMI to the local application. The NTMS is used to configure and manage the Trusted Time products.

The TTI is designed so that trust in the distributed time value is successfully propagated from the NMI down through all intermediate distribution layers to the application itself. Individual trusted time products, such as the National Measurement Institute Trusted Time Server (NMIServer), the Trusted Master Clock (TMC), and the TT StampServer (TSS), are designed to be tamper resistant per FIPS 140-1 recommendations. Trusted time components comprised of multiple systems, such as the Network Time Management System (NTMS) and the Certification Authority (CA), are hosted in physically secure facilities. The CA employs tamper protection to prevent disclosure of its root private signature key. Audit records certifying the clock calibration, traceable back to the NMI, are available for all layers of the TTI. The trust model also is created and maintained by audited operational procedures. The audit of the NTMS and CA is performed by an external third party such as a large accounting firm.

If the TTI is used to also perform time stamping, the requester of the time stamp, not the TTI, is responsible for verifying the correctness of the time stamp produced by the TTI.

TTI System Architecture

The Datum TTI is comprised of a number of distinct elements networked together in order to create and distribute trusted time. These elements include: the NMI timing source, the NMI Trusted Time Server (NMIServer), a Certification Authority (CA) and its related Public Key Trusted Time API (TTAPI) embedded in the application, and the TTI Network Time Management System (NTMS). A general view of the TTI system architecture is shown in the figure on the next page.

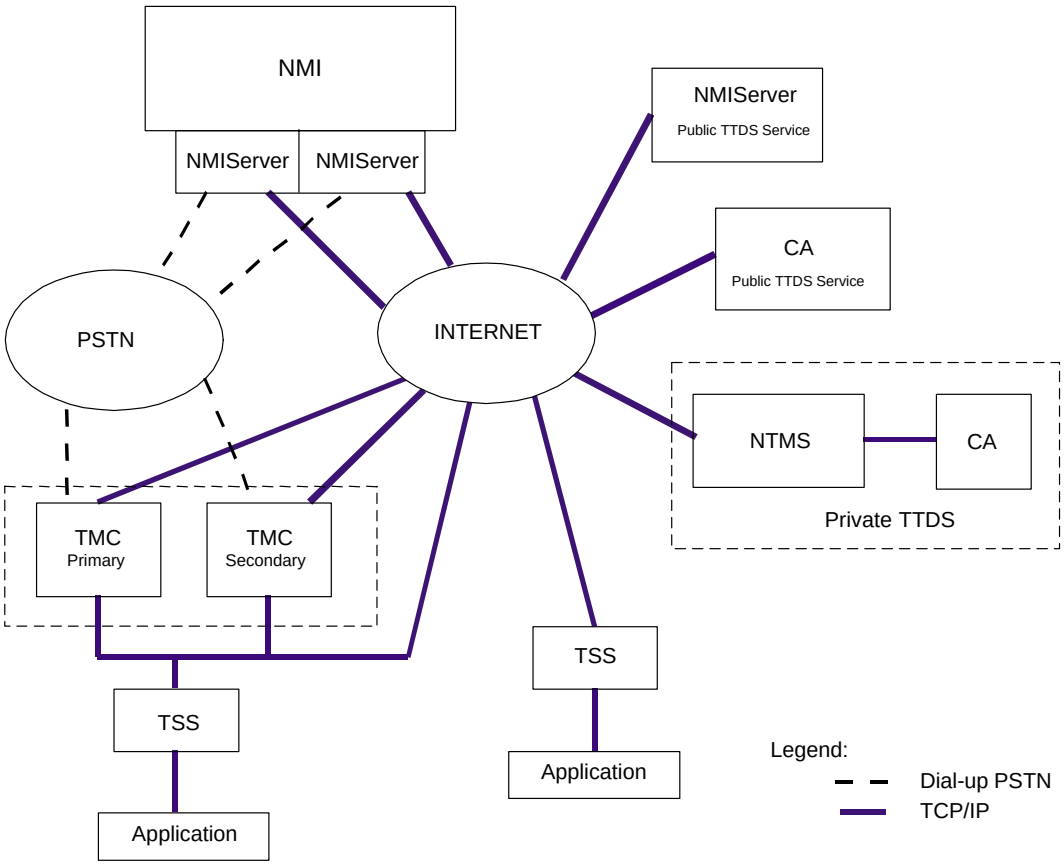

Figure 2-1: TT System Architecture

Creating Trusted Time

Trusted Time is created at the National Measurement Institute using the NMIServer-7000 which is a standalone device. The NMIServer is responsible for measuring the clock offsets of the Trusted Time Distribution System (TTDS) Stratum 1 TMC units (see Figure 2). Initial versions of the NMIServer will perform this measurement only over a dedicated phone line. The NMI is responsible for monitoring the accuracy of the NMIServer-produced time and the operation of the NMIServer itself.

To measure the clock offsets of the Stratum 1 TMCs as well as those of other non-Datum clocks, the NMIServer supports the Datum Secure NTP (DS/NTP) protocol. The protocol is based upon the Secure NTP protocol under development in the IETF.

Timing Source

For its timing source, the NMIServer-7000 accepts and externally supplied timing signal. This is typically a one pulse-per-second (PPS) and a 10MHz frequency reference. In the United States, NIST supplies timing to the NMIServer. Note that when an external timing source is used, it should be directly cabled to the NMIServer. For tamper protection, an intervening network should not be allowed.

Time Distribution

Following the TTDS Stratum concept, Levels 1, 2, and 2A are each implemented using TMCs which are standalone devices. Each of these Levels is comprised of at least two TMC units. Each TMC has a set of lower TMCs that it is responsible for time certifying. These sets of lower TMCs are assigned by the NTMS. The NTMS ensures that two upper TMCs are assigned to each lower TMC. This structure ensures sufficient redundancy so that the failure of a single higher TMC does not affect the operation and trust of the lower TMCs. Of course, it is possible to operate a TMC in a standalone configuration but this is not a recommended configuration.

For TTDS Stratum 2A (or 2 if 2A is not used), each TMC are assigned a set of TTSs (TT StampServers) by the NTMS for which it is responsible for time certifying. The NTMS ensures that two TMCs are assigned to each set of TSSs in order to provide redundancy for the time certification process. The TMC uses DS/NTP and UDP/IP to access each of its assigned TLCs periodically to measure its time offset, and, if everything is normal, certify the TSS to be within a certain offset from NMI UTC. It may also send small time corrections to the TSS which can be used to make adjustments to the TSS clock to keep it within specification. If a TMC finds that a TSS has a recent time certificate, then it takes no action; otherwise, it performs a time certification of that TSS. Note that the TMCs must have an Ethernet TCP/IP connection in order to communicate with the NTMS.

Trusted Time StampServer

The TSSs operate as Time Stamp Authority (TSA) “engines” and can be used by TSAs to implement time stamping as specified in the IETF PKIX Time Stamp document [TS]. The TSSs provide trusted time, in the form of a Temporal Token (DTT) which may be sent directly to the TSA or embedded in a Time Stamp Token (TST) and then passed to the TSA. The TSS is designed to be hosted in a customer-owned server. The TSS has its own Ethernet TCP/IP connection for communications with the TMC and NTMS.

Client applications access the TSSs using the Trusted Time API (TTAPI).

NTMS: Managing TTP Elements

The Network Time Management System (NTMS) includes an element management system which has the ability to remotely configure and monitor the NMIServer, TMCs, and TSSs using a modified version of SNMPv1 as a secure management protocol. In addition this management system includes a central database which is used to store all TMC and TSS certification logs. Since the NTMS knows all TTP elements comprising the TTI, it acts as a Registration Authority (RA) to the Certification Authority (CA) for the issuance of certificates to the TTP elements. The NTMS may also host an optional billing system for all TTDS subscribers.

The NTMS’ design is highly scalable so that it can form the core of a network operations center for a commercial TTDS service. Note that for such a commercial TTDS service, the commercial (or public) TTDS monitors only the TTDS elements under its direct control. If an OEM has a private TTDS that subscribes to the commercial service, the commercial NTMS monitors only the OEM’s TMCs which are

directly calibrated by the TTDS Stratum 1 TMC units. OEMs must purchase and operate a NTMS from Datum in order to securely monitor and control their private labeled units.

TTDS customers may use their SNMPv1 management equipment to monitor their TTDS components, but they may not remotely control them.

Certification Authority

The certification authority (CA) may be provided by either a CA service or a CA product that is integrated into the NTMS. Using the IETF PKIX protocols, the TTP units request certificates from the NTMS RA for their public keys. The NTMS RA ensures that only authorized TTP elements receive certificates from the CA. The CA is also responsible for distributing certificate revocation lists to the TTP deployed in the TTI.

Timestamping Function of the TSS

The primary function of the TT StampServer is to issue the Datum TSS issues signed timestamps in the form of time stamp Tokens (TST). The time stamp Token is defined by PKIX Timestamping Protocols (please refer to the appendices of this document for more detailed information). TSTs issued by a TSS conform to the IETF TST definition with the addition of the [TBD] extension.

In addition to timestamping, the TSS can provide the current time value in an unsigned, raw format.

TSS Key Management

The TSS requires several different key pairs to operate. These key pairs are used to sign Time Stamps and to perform secure operations with the TTI. For a detailed description of the TTI public key and certificate model, please read Appendix A and B.

To facilitate public key management, the TTAPI library provides functions which allow the user to:

- Generate Time Stamp Authority (TSA) Signing Keys
- Generate TSA Certificate Requests
- Import/Export TSA Certificates
- Generate TTDS1 and TTDS2 Signing Keys

The *Datum TTI System Definition* defines three different keys that might be used on a TSS. These keys are identified in the SDK with the TTAPI_KEY enumeration:

```
enum TTAPI_KEY
{
KEY_PAIR_TSA,
KEY_PAIR_TTDS1,
KEY_PAIR_TTDS2
};
```

In addition, the TSS supports DSA key types. These key types are distinguished in the key generation function with the TTAPI_KEY_TYPE enumeration:

```
enum TTAPI_KEY_TYPE
{
KEY_TYPE_DSA,
KEY_TYPE_RSA
};
```

Notes

Chapter 2: Getting Started

This chapter provides instructions for installing the Trusted Time StampServer hardware, SDK, and software demo.

Installing the TSS Hardware

Following are the instructions for installing the Trusted Time StampServer hardware,

NOTE: The Datum TSS hardware should be installed before the SDK is installed.

To install the TLC PCI board:

1. Turn off power to the PC.
2. Pick a vacant PCI slot in your PC.
3. Using ESD preventive wrist strap, take the TSS PCI (TT StampServer module) card from its wrapping and place it in the slot in your PC's mother board.
4. Slide the PCI board in carefully to the slots orienting it.
5. Snap the holding clips into place.
6. Replace your PC enclosure.
7. Connect the card to the Network via an Ethernet connection.
8. Power up your system.
9. Install the device driver (see below).
10. Consult the user documentation that came with your workstation for specific PCI board installation instructions.

Installing the Software Development Kit

When the TLC PCI board is installed, you may proceed with the installation of the Software Development Kit (SDK).

To install the TTAPI SDK:

1. Insert the CD labeled “Trusted Time StampServer” into your CD-ROM drive.
2. The `autorun` should start in a few seconds. Follow the installation instructions on your screen. This program installs the driver for the Datum TSS and installs all the SDK files on your hard drive.
3. Run the **`TSADemo.exe`** program to verify installation (please refer to the section below, “Software Programs”).

The following directories are created during the software installation process:

Table 2-1: Directories Created

Directory	What it contains
include	The include files that define the TTAPI functions
bin	TTAPI DLLs
lib	Library files for linking to TTAPI functions
doc	This document
src	Sample source code

Software Programs

A Demo Program (**TSADemo.EXE**) is included with the TLC PCI module.

The TSADemo.EXE program shows how timestamps are issued. The program prompts the user to log in and select a file to be timestamped. Once this information is entered, the time stamp is issued and output to a file.

Chapter 3: API Functions, Categories, and Definitions

Trusted Time API Overview

This section defines the Trusted Time API and lists specific functions.

The TTAPI SDK is a powerful set of API functions that allows developers to:

- Administer the TSS
- Request Timestamps from a TSS
- Decode timestamps

API Functions, Categories, and Definitions

The API functions are divided into four categories depending upon the tasks they must perform:

- Session Functions
  Used to manage TSS sessions and to initialize the API library
- Supervisory Functions
  Used to perform supervisory operations, such as TSS initialization and key generation.
- Operational Functions
  Used to perform timestamping issuing and verification services.
- Utility Functions
  Used to manipulate ASN.1 types and perform other miscellaneous operations.

TTAPI Functions by Category

These are the TTAPI functions.

Table 3-1: TTAPI Functions by Category

Category	Method
Session	*TTAPI_Open Session*
	TTAPI_CloseSession
	TTAPI_Login
	TTAPI_Logout
Supervisory	*TTAPI_S_Initialize*
	TTAPI_S_ChangePIN
	TTAPI_S_GenerateKeyPair
	TTAPI_S_GenerateCertRequest
	TTAPI_S_ImportCertificate
	TTAPI_S_ExportPublicKey
	TTAPI_S_SetParameter
	TTAPI_S_GetParameter
Operational	*TTAPI_S_SelfTest*
	TTAPI_IssueTimestampToken
	TTAPI_GetToken
	TTAPI_ReleaseToken
	TTAPI_GetCertificate
	TTAPI_GetTCCert
	TTAPI_GetTime
Utility	*TTAPI_DecodeTimestampToken*
	TTAPI_DecodeTCCert
	TTAPI_EncodeTimestampRequest

Session Functions

Before any TSS services can performed, a TSS session must be created. Use the *TTAPI_OpenSession* function to open a session with the TSS. This process helps insure the TSS is installed and the SDK can communicate with it. The handle that is returned in the *TTAPI_OpenSession* function is used in the other TTAPI functions that require a session handle.

Once a TSS session has been created, any non-supervisory functions can be accessed. The *TTAPI_Login* function authenticates a session for supervisory access to the TSS. This login function must return

successfully before supervisory functions can be called. When logged in as a supervisor, the session owner can use both the supervisor and time stamping functions.

Applications should call the *TTAPI_Logout* function and the *TTAPI_CloseSession* function upon completion of time stamping and supervisory services.

Supervisory Functions

The Trusted Time StampServer (TSS) is administered using the TTAPI supervisory functions. Specifically, supervisory functions allow applications to control the configuration of a TSS. For a list of session functions, refer to the previous sections, "TTAPI Functions by Category" and "Session Functions," in this chapter.

These functions allow the application to:

- Manage TSS Keys
- Set TSS Configuration Parameters
- Manage the TSS PIN

Applications must login to the TSS as an Administrator before any Supervisory functions can be called.

Warning: Applications developers should be aware that misuse and incorrect use of these functions could cause the TSS to stop operating in a TTI system. Functions for initializing the TSS, generating key pairs, and setting parameters can be destructive if not used properly. Application designers should have a thorough understanding of the Cryptographic Key Architecture of the Datum TTI System before deploying their TSS applications. Please refer to *Appendix B* for more information.

Operational Functions

Operational functions are used to perform time stamp issuing and verification services. The *TTAPI_IssueTimeStampToken* function is used to request time stamps from the TSS. Upon success, this function returns a handle to the resulting time stamp. The encoded, signed time stamps are retrieved using the function, *TTAPI_GetTimeStamp.*

Time Stamps that are produced by the TSS are signed and ASN.1 encoded. The *TTAPI_IssueTimeStampToken* function issues a TimeStampToken (TST) PDU. This data can be decoded using the TTAPI Utility functions.

Utility Functions

Also provided with the TTAPI SDK is a set of utility functions that assist in developing time stamping functions. These functions provide the following features:

- ASN.1 Time Stamp Token Decoding
- ASN.1 Time Stamp Request Encoding

The utility functions are maintained in a separate library from the other TTAPI functions. This separate library allows these functions to be included in applications that may not be connected to a TSS (that is, Time Stamp Client application).

The TTAPI Utility functions are located in the `TTAPIUtils.DLL` file and the functions can be linked into your program with the `TTAPIUtils.lib` file.

Chapter 4: API Specifications

Session Functions

TTAPI_OpenSession

int ***TTAPI_OpenSession*** *(TTAPI_SESSION_HANDLE *phTSSSession);*

Description:
In order to communicate with the TSS, a session must be established. The *TTAPI_OpenSession* function is used to open a session and the *TSSCloseSession* function closes a session.

This function must be called before any other call to the TTAPI is made. This function initializes communications with the TSS and prepares the TTAPI and TSS for time stamping services. This call returns a session handle that must be used in all other TTAPI calls.

Parameters:
*TTAPI_SESSION_HANDLE *phTSSSession*

This parameter points to the location where the TSS Session handle is placed. This handle is used in all other TTAPI calls to indicate the session to use for a particular operation.

Return Values:
TTAPI_SUCCESS

TTAPI_CloseSession

int ***TTAPI_CloseSession*** *(TTAPI_SESSION_HANDLE hTSSSession);*

This function closes a TSS session and terminates the connection between the API and TSS. If the user is logged in when this function is called, that user is logged out before the session is cleaned up. It is not necessary to explicitly call *TTAPI_Logout* function before *TTAPI_CloseSession.*

Parameters:
TCL_SESSION_HANDLE hTSSSession

This parameter is the handle of the session that was returned by *TTAPI_OpenSession*.

Return Values:
TTAPI_SUCCESS

TTAPI_INVALID_SESSION_HANDLE

TTAPI_Login

*int **TTAPI_Login** (TTAPI_SESSION_HANDLE hTSSSession, char *szPassword);*

This function logs a supervisor into the TSS. The *TTAPI_OpenSession* function must be called before this function can be called.

Parameters:
TTAPI_SESSION_HANDLE hTSSSession

This parameter is the handle of the session and must be obtained from the *TTAPI_OpenSession* function.

*char *szPassword*

This parameter contains the password for the TSS.

Return Values:
TTAPI_SUCCESS

TTAPI_INVALID_SESSION_HANDLE

TTAPI_INVALID_PASSWORD

TTAPI_NOT_INITIALIZED

TTAPI_Logout

*int **TTAPI_Logout** (TTAPI_SESSION_HANDLE hTSSSession);*

This function logs a user out of the TSS.

Parameters:
TTAPI_SESSION_HANDLE hTSSSession

This parameter is the handle of the session and must be obtained from the *TTAPI_OpenSession* function.

Return Values:
TTAPI_SUCCESS

TTAPI_INVALID_SESSION_HANDLE

TTAPI_INVALID_PASSWORD

TTAPI_NOT_INITIALIZED

Supervisory Functions

Supervisory functions can only be performed if a user is logged in as a supervisor.

TTAPI_S_Initialize

*int **TTAPI_S_Initialize** (TTAPI_SESSION_HANDLE hTSSSession,*

*char *szSupervisorPassword)*

This function initializes the TSS. If the TSS has been previously initialized, this function destroys the settings on the TSS, including the TSS public/private key pairs and certificate. The following operations are performed during TSS initialization:

- Cryptographic storage is initialized
- TSS password supervisor set
- TSS Configuration Parameters are reset

This function destroys any keys and configuration settings that exist on the TSS when the function is called.

Parameters:
TTAPI_SESSION_HANDLE hTSSSession

This parameter is the handle of the session and must be obtained from the *TTAPI_OpenSession* function.

*char *szSupervisorPassword*

Contains the new supervisor password for the TSS.

Return Values:
TTAPI_SUCCESS

TTAPI_INVALID_SESSION_HANDLE

TTAPI_INVALID_PASSWORD

TTAPI_NOT_INITIALIZED

TTAPI_S_ChangePIN

*int **TTAPI_S_ChangePIN**(TTAPI_SESSION_HANDLE *phTSSSession,*

*char *szNewPIN)*

Description:
This function is used to set the supervisor PIN on the TSS.

Parameters:
TTAPI_SESSION_HANDLE *phTSSSession

This parameter points to the location where the TSS Session handle is placed. This handle is used in all other TTAPI calls to indicate the session to use for a particular operation.

*char *szNewPIN*

Contains the new PIN for the TSS supervisor.

Return Values:
TTAPI_SUCCESS

TTAPI_S_ExportPublicKey

int ***TTAPI_S_ExportPublicKey****(TTAPI_SESSION_HANDLE hTSSSession,*
TTAPI_KEY eKey,
*unsigned char *pKeyBuffer,*
*int *piRequestBufferLength)*

This function exports the specified TSS public key. The key is exported as an ASN.1-encoded SubectPublicKeyInfo PDU.

Parameters:

TTAPI_SESSION_HANDLE hTSSSession
This parameter is the handle of the session and must be obtained from the *TTAPI_OpenSession* function.

TTAPI_KEY eKey
This parameter identifies the key to export.

*unsigned char *pKeyBuffer*
This parameter must point to a buffer that can receive the encoded public key value.

*int *piRequestBufferLength*
On input, this parameter must contain the size of the buffer pointed to by *pKeyBuffer*. On output, this value is set to the size of the encoded key.

Return Values:
TTAPI_SUCCESS
TTAPI_INVALID_SESSION_HANDLE
TTAPI_INVALID_PASSWORD
TTAPI_NOT_INITIALIZED

TTAPI_S_GenerateCertRequest

int ***TTAPI_S_GenerateCertRequest****(TTAPI_SESSION_HANDLE hTSSSession,*
TTAPI_KEY eKey,
TTAPI_CERT_REQ_TYPE eCertReqType,
*unsigned char *pRequest,*
*int *piReqBufferLength)*

This function causes the TSS to create a certificate request message for the specified key. Two types of certificate requests are supported: PKCS10 and CRM.

Parameters:
TTAPI_SESSION_HANDLE hTSSSession

This parameter is the handle of the session and must be obtained from the *TTAPI_OpenSession* function.

TTAPI_KEY eKey
This parameter identifies the key pair to use to generate the request.

TTAPI_CERT_REQ_TYPE eCertReqType
This parameter identifies the type of certificate request to issue.

enum TTAPI_CERT_REQ_TYPE
{
CERT_REQ_PKCS10,
CERT_REQ_CRM
};

*unsigned char *pRequest*
This parameter must point to a buffer where the encoded request will be copied.

*int *piReqBufferLength*
On input, this parameter must contain the size of the buffer pointed to by *pRequest*. On output, this value is set to the size of the encoded request.

Return Values:
TTAPI_SUCCESS
TTAPI_INVALID_SESSION_HANDLE
TTAPI_INVALID_PASSWORD
TTAPI_NOT_INITIALIZ

TTAPI_S_ImportCertificate

*int **TTAPI_S_ImportCertificate**(TTAPI_SESSION_HANDLE hTSSSession,*
TTAPI_KEY eKey,
*unsigned char *pCertificate,*
int iCertificateLength)

DELETE?
This function is used to import a certificate that might have been issued by an external Certificate Authority. The public key in the certificate must match the public key of the key pair specified by *eKey*. This function accepts ASN.1-encoded X.509 certificates.

Parameters:
TTAPI_SESSION_HANDLE hTSSSession

This parameter is the handle of the session and must be obtained from the *TTAPI_OpenSession* function.

TTAPI_KEY eKey
This parameter identifies the key pair with which the imported certificate is associated.

*unsigned char *pCertificate*
This parameter must point to a buffer where the encoded request will be copied.

int iCertificateLength
This parameter is used to specify the size of the certificate in *pCertificate*.

Return Values:
TTAPI_SUCCESS

TTAPI_KEY_MISMATCH

TTAPI_INVALID_SESSION_HANDLE

TTAPI_INVALID_PASSWORD

TTAPI_NOT_INITIALIZED

TTAPI_S_GenerateKeyPair

int ***TTAPI_S_GenerateKeyPair****(TTAPI_SESSION_HANDLE hTSSSession,*
TTAPI_KEY_TYPE eType,
TTAPI_KEY eKey)

This function is used to tell the TSS to generate one of its key pairs. The new key pair is of the type specified in the *eType* parameter. The *eKey* parameter identifies the key to generate. If a key pair exists in the TSS for the specified key, it is replaced by the key pair. Additionally, if a certificate exists for the key, it is erased. The size of the generated keys depends upon the type of key generated. Please refer to Appendix B for key sizes.

Does request go to NTMS? If not it might not be supported.

Parameters:

TTAPI_SESSION_HANDLE hTSSSession
This parameter is the handle of the session and must be obtained from the *TTAPI_OpenSession* function.

TTAPI_KEY_TYPE eType
This parameter specifies the type for the new key.

enum TTAPI_KEY_TYPE

{

KEY_TYPE_DSA,

KEY_TYPE_RSA

};

TTAPI_KEY eKey
This parameter specifies the TSS key pair to generate.

```
enum TTAPI_KEY
{
KEY_PAIR_TSA,
KEY_PAIR_TTDS1,
KEY_PAIR_TTDS2
};
```

Return Values:
TTAPI_SUCCESS

TTAPI_INVALID_SESSION_HANDLE

TTAPI_INVALID_PASSWORD

TTAPI_NOT_INITIALIZED

TTAPI_S_SetParameter

int ***TTAPI_S_SetParameter*** *(TTAPI_SESSION_HANDLE hTSSSession,*
*char *szParameterName,*
*BYTE *pParameterValue,*
int iParameterLength)

This function sets the value of a TSS configuration parameter. (Please see "TSS Configuration Parameters" in this document). If the TSS's state is *TSSSTATE_UNINITIALIZED*, then this function can be called immediately after the *TTAPI_OpenSession* function. If the TSS is initialized, then the user must call *TTAPI_Login* before this function can be called.

Parameters:

TTAPI_SESSION_HANDLE hTSSSession
This parameter is the handle of the session and must be obtained from the *TTAPI_OpenSession* function.

*char *szParameterName*
Contains the name of the TSS parameter to set.

*BYTE *pParameterValue*
Contains the new value for this parameter.

int iParameterLength
Contains the length of the value pointed to by *pParameterValue*.

Return Values:
TTAPI_SUCCESS

TTAPI_INVALID_SESSION_HANDLE

TTAPI_INVALID_PASSWORD

TTAPI_NOT_INITIALIZED

TTAPI_S_GetParameter

int ***TTAPI_S_GetParameter*** *(TTAPI_SESSION_HANDLE hTSSSession,*
*char *szParameterName,*
*BYTE *pParameterValue,*
*int *piParameterLength)*

This function obtains the value of a TSS configuration parameter (Please refer to "TSS Configuration Parameters").

Parameters:

TTAPI_SESSION_HANDLE hTSSSession
This parameter is the handle of the session and must be obtained from the *TTAPI_OpenSession* function.

*char *szParameterName*
Contains the name of the TSS parameter to get

*BYTE *pParameterValue*
The buffer that will receive this parameter value.

*int *piParameterLength*
On input, this value must contain the length of the buffer pointed to by *pParameterValue*. On output, this value is set to the actual length of the parameter. If the buffer is too small for the parameter, the return code is *TTAPI_BUFFER_TOO_SMALL* and the actual parameter's length is still returned in this *piParameterLength.*

Return Values:
TTAPI_SUCCESS
TTAPI_INVALID_SESSION_HANDLE
TTAPI_INVALID_PASSWORD
TTAPI_NOT_INITIALIZED
TTAPI_BUFFER_TOO_SMALL

TTAPI_S_SelfTest

int ***TTAPI_S_SelfTest****(TTAPI_SESSION_HANDLE hTSSSession)*

This function tells the TSS to perform a self-test. The parameters of this function are [tbd].

Parameters:

TTAPI_SESSION_HANDLE hTSSSession
This parameter is the handle of the session and must be obtained from the *TTAPI_OpenSession* function.

Return Values:
TTAPI_SUCCESS
TTAPI_INVALID_SESSION_HANDLE

Operational Functions

The following functions can be used to perform time stamping operations.

- TTAPI_IssueTimestampToken
- TTAPI_GetToken
- TTAPI_GetTime
- TTAPI_ReleaseToken
- TTAPI_GetCertificate

The above functions all require that a TTAPI session be open and a user logged into the device. The time stamp issuing functions do not return a time stamp. Because time is of the essence, it is critical that time stamps be issued when a request is made. Therefore, the issuing functions do not manage the buffers that are used to store time stamps. Time stamps are retrieved using the *TTAPI_GetToken* function.

TTAPI_IssueTimestampToken

int ***TTAPI_IssuzeTimestampToken****(TTAPI_SESSION_HANDLE hTSSSession,*
*unsigned char *pEncodedRequest,*
int iEncodedRequestSize,
*PTTAPI_TokenHandle *ppTokenHandle)*

This function issues a Time Stamp Token (TST).

Parameters:

TTAPI_SESSION_HANDLE hTSSSession
This parameter is the handle of the session and must be obtained from the *TTAPI_OpenSession* function.

*unsigned char *pEncodedRequest*
This parameter must point to a buffer containing an ASN.1-encoded TimeStampReq.

int iEncodedRequestSize
Size of the encoded TimeStampReq pointed to by *pEncodedRequest*.

*PTTAPI_TokenHandle *ppTokenHandle*
Points to a *TTAPI_TokenHandle* and receives the handle of the newly generated token. *TTAPI_GetToken* can be called to retrieve the token.

Return Values:
TTAPI_SUCCESS

TTAPI_GetToken

int ***TTTAPI_GetToken*** *(TTAPI_SESSION_HANDLE hTSSSession,*
TTAPI_TokenHandle hToken,
*BYTE *pToken,*
*ULONG *pulTokenLength,*
BOOL bRelease)

This function retrieves a token from the TSS. What kind of token?

Parameters:

TTAPI_SESSION_HANDLE hTSSSession
This parameter is the handle of the session and must be obtained from the *TTAPI_OpenSession* function.

TTAPI_TokenHandle hToken
This parameter specifies the handle of the token to get.

*BYTE *pToken*
Points to a buffer that will receive the encoded token.

*ULONG *pulTokenLength*
On input, this must point to an integer that contains the length of the buffer pointed to by *pToken*. On output, even if the buffer is too small, this value is set to the length of the token.

BOOL bRelease
If *TRUE*, the handle for this token is released and the token is destroyed. Note that the *TTAPI_ReleaseToken* can also be used to free a token.

Return Values:
TTAPI_SUCCESS

TTAPI_BUFER_TOO_SMALL

TTAPI_GetTime

int **TTTAPI_GetTime** (TTAPI_SESSION_HANDLE hTSSSession, unsigned long *pulTime)

This function gets the unsigned, raw time from the TSS. The time is returned as [TBD].

Parameters:
TTAPI_SESSION_HANDLE hTSSSession

This parameter is the handle of the session and must be obtained from the *TTAPI_OpenSession* function.

Return Values:
TTAPI_SUCCESS

TTAPI_ReleaseToken

int ***TTAPI_ReleaseToken*** *(TTAPI_SESSION_HANDLE hTSSSession ,*
TTAPI_TokenHandle hToken)

This releases a token handle and frees all resources associated with storage of the token.

Parameters:

TTAPI_SESSION_HANDLE hTSSSession
This parameter is the handle of the session and must be obtained from the *TTAPI_OpenSession* function.

TTAPI_TokenHandle hToken
This value specifies the handle of the token to release.

Return Values:
TTAPI_SUCCESS

TTAPI_INVALID_HANDLE

TTAPI_GetCertificate

*int **TTAPI_GetCertificate** (TTAPI_TokenHandle hToken,*
TTAPI_KEY eKey,
BOOL bGetChain,
*unsigned char *pEncodedCert,*
*int *piCertBufferLength)*

This function retrieves a certificate from the TSS for the specified key. This function can return either a certificate or a complete certificate chain that contains all of the certificate up to and including the CA certificate. In both cases, the certificate(s) that are returned are ASN.1-encoded.

Parameters:

TTAPI_SESSION_HANDLE hTSSSession
This parameter is the handle of the session and must be obtained from the *TTAPI_OpenSession* function.

TTAPI_KEY eKey
This parameter specifies the key of the certificate to retrieve.

BOOL bGetChain
If this parameter is TRUE, then the complete certificate chain is returned. If FALSE, then only the certificate for the specified key is returned.

*unsigned char *pEncodedCert*
This parameter must point to a data buffer than can receive the encoded certificates.

*int *piCertBufferLength*
On input, this parameter must contain the size of the buffer pointed to by *pEncodedCert*. On output, this value is set to the size of the encoded certificates.

Return Values:
TTAPI_SUCCESS

TTAPI_INVALID_HANDLE

TTAPI_GetTCCert

*int **TTAPI_GetTCCert** (TTAPI_TokenHandle hToken,*
*unsigned char *pEncodedCert,*
*int *piCertBufferLength)*

This function retrieves the current TTCert from the TSS.

Parameters:

TTAPI_SESSION_HANDLE hTSSSession
This parameter is the handle of the session and must be obtained from the *TTAPI_OpenSession* function.

*unsigned char *pEncodedCert*
This parameter must point to a data buffer than can receive the encoded certificate.

*int *piCertBufferLength*
On input, this parameter must contain the size of the buffer pointed to by *pEncodedCert*. On output, this value is set to the size of the encoded certificate.

Return Values:
TTAPI_SUCCESS

TTAPI_INVALID_HANDLE

Utility Functions

The TTAPI utility functions are provided to assist developers in creating complete applications with the TTAPI SDK. These functions are provided in a separate library to make it easier to integrate these functions into applications that might request only tokens or decoded tokens, such as a TST client application. The name of this library is TTAPIUtils.

The TTAPI utility functions do not require an open TTAPI session.

TTAPI_DecodeTimestampToken

*int **TTAPI_DecodeTimestampToken**(BYTE * pEncodedToken,*
ULONG ulTokenLength,
*TST *pToken)*

This function decodes an encoded TST. The passed-in structure is filled with the contents of the encoded token.

Parameters:

*BYTE *pEncodedToken*
Points to the encoded token.

ULONG ulTokenLength
Contains the length of the encoded token.

*TST *pToken*
This parameter must point to a structure that receives data from the encoded token. Please refer to the *TST* definition under Structure Definitions.

Return Values:
TTAPI_SUCCESS

TTAPI_ASN1_ERROR

TTAPI_DecodeTCCert

*int **TTAPI_DecodeTCCert(**BYTE * pEncodedCert,*
ULONG ulCertLength,
*TCCERT *pCert)*

This function decodes an encoded TCCERT. The passed-in structure is filled with the contents of the encoded certificate.

Parameters:

*BYTE *pEncodedCert*
Points to the encoded token.

ULONG ulTokenLength
This parameter must be set to the length of the encoded certificate.

*TCCERT *pCert*
This parameter must point to a structure that receives the data from the encoded certificate. Please refer to the *TCCERT* definition under Structure Definitions.

Return Values:
TTAPI_SUCCESS

TTAPI_ASN1_ERROR

TTAPI_EncodeTimestampRequest

*int **TTAPI_EncodeTimestampRequest**(TSR *pRequest,*
*unsigned char *pEncodedRequestt,*
*int *piRequestBufferLength);*

This function ASN.1 encodes a Time Stamp Request (TSR).

Parameters:

*TSR *pRequest*
The TSRequest structure contains all of the information that can go into a Time Stamp Request. This parameter should point to a TSRequest structure that contains the information that goes into the request.

*unsigned char *pEncodedRequest*
This parameter should point to a buffer that receives the encoded request.

*int *piRequestBufferLength*
On input, this parameter must be set to the size of the buffer that is pointed to by *pEncodedRequest*. On output, this value contains the size of the encoded request.

Return Values:
TTAPI_SUCCESS

TTAPI_ASN1_ERROR

TSS Configuration Parameters

Subject Name

define SZ_PARAM_SUBJECT_NAME "Subject Name"

This parameter sets the TSS's name. This name is used as the TSS's *subject* in the Time Stamp Token and other signed output.

IP Address

define SZ_PARAM_IP_ADDRESS "IP Address"

Use the IP Address parameter to set the IP Address of the TSS. This parameter is a string i.e. "209.214.56.4").

Subnet Mask

define SZ_PARAM_SUBNET_MASK "Subnet Mask"

Use the Subnet mask parameter to set the subnet mask of the TSS. This parameter is a string (i.e. "255.255.255.0").

Gateway

define SZ_PARAM_GATEWAY "Gateway"

Use this parameter to set the default gateway for the TSS. This parameter is a string (i.e. "204.56.43.1").

Sample Applications

SimpleTSA

The Simple TSA program is a simple example of how timestamps can be issued using the API. The program prompts the user to login and to select a file to be timestamped. Once this information is entered, the time stamp is issued and output to a file.

```
//**************************************************************************
//
//   SimpleTSA.cpp
//
//   This program demonstrates the functionality provided through TTAPI. It
//   is provided for demonstration purposes only. Any part of this file may
//   be reused or copied freely.
//
//   This product includes cryptographic software
//   written by Eric Young (eay@cryptsoft.com)
//
//**************************************************************************

include "stdafx.h"
include <conio.h>
include <ctype.h>
include <stdio.h>
include "Ttapi.h"
include "TtapiTypes.h"
include "sha.h"
include "sha_local.h"


int main(int argc, char* argv[])
{
   char  szPassword[128];
   TTAPI_SESSION_HANDLE hTTAPISession;
   TimeStampRequest tsrTimeStampReq;
   TTAPITokenHandle hTokenHandle;
   unsigned char ucEncodedTSR[1024];
```

```
int iEncodedReqLen = sizeof(ucEncodedTSR);
int itoken, numwritten;
unsigned char ucOutputBuffer[1024];
unsigned long iTokenLen;
char cOutput;
char  szFileName[256];
FILE  *pFile;

//
// Prompt the user for the PIN
//
printf("Enter password: ");
scanf("%s", szPassword);

//
// Open a session with the TSS
//
if (TTAPI_OpenSession(&hTTAPISession) != TTAPI_SUCCESS)
{
   printf("\r\nTTAPI_OpenSession Failed");
   return -1;
}

//
// Get the name of the file to time stamp and open it
//
printf( "\nEnter file name: ");
scanf( "%s", szFileName);
if( (pFile = fopen( szFileName, "rb" )) == NULL )
{
   printf( "Error opening the file\n" );
   TTAPI_CloseSession(hTTAPISession);
   return -1;
}

//
```

```
// Fill in the time stamp request structure
//
tsrTimeStampReq.iVersion = 1;
tsrTimeStampReq.policy = 0;
tsrTimeStampReq.iNonce = 10;

//
// Calculate hash value of the file
//
unsigned char szBuffer[1024];
int numread = 512;
SHA_CTX c;
SHA1_Init(&c);
while (numread == 512)
{
   numread = fread( szBuffer, 1, 512, pFile );
   printf( "Number of items read = %d\n", numread );
   SHA1_Update(&c, szBuffer, numread);
}
SHA1_Final(tsrTimeStampReq.hashVal, &c);
tsrTimeStampReq.iHashLen = 20;

//
// We can close the file now
//
fclose( pFile );

//
// Encode the time stamp Request
//
if (TTAPI_EncodeTimestampRequest(hTTAPISession,
                        &tsrTimeStampReq,
                        ucEncodedTSR,
                        &iEncodedReqLen) != TTAPI_SUCCESS)
{
   printf("Problem encoding time stamp Request\n");
```

```
    TTAPI_CloseSession(hTTAPISession);
    return -1;
}

//
// ISSUE time stamp
//
if (TTAPI_IssueTimestampToken(hTTAPISession,
                        ucEncodedTSR,
                        iEncodedReqLen,
                        &hTokenHandle) != TTAPI_SUCCESS)
{
    printf( "Problem issuing time stamp\n" );
    TTAPI_CloseSession(hTTAPISession);
    return -1;
}

//
//  Get the time stamp and output it to a file
//
printf("\nTimestamp Successful\n\n");
printf("Output time stamp to a file? (y/n): ");
do
{
    cOutput = _getch();
    cOutput = toupper(cOutput);
    printf("\n\n");
} while((cOutput != 'Y') && (cOutput != 'N'));

if (cOutput == 'Y')
{
    //
    // get the name of the file to write to
    //
    printf("Enter file name: ");
    scanf("%s", szFileName);
```

```
//
// Get time stamp and write it to a file.
//
if( (pFile = fopen( szFileName, "wb" )) == NULL )
{
   printf("File open failed\n");
}
else
{
   //GET TOKEN
   if (pFile != NULL)
   {
      iTokenLen = sizeof(ucOutputBuffer);

      itoken = TTAPI_GetToken(hTTAPISession,
                        hTokenHandle,
                        ucOutputBuffer,
                        &iTokenLen, FALSE);

      if (itoken == TTAPI_SUCCESS)
      {
         //WRITE TOKEN TO FILE
         numwritten = fwrite(ucOutputBuffer, 1, iTokenLen, pFile);
         printf( "Wrote %d bytes\n", numwritten );
      }
      else if (itoken = TTAPI_BUFFER_TOO_SMALL)
      {
         printf("Buffer too small\n");
         printf("Buffer size is %d bytes\n", iTokenLen);
      }
      else
      {
         printf("Problem writing time stamp to file\n");
      }
   }
```

```
            fclose(pFile);
        }
    }


    //Close session with the TSS
    TTAPI_CloseSession(hTTAPISession);

    return 0;
}
```

Chapter 5: Trusted Time Functions

This chapter provides details about:

- Temporal Token Validation
- Registration Authority (RA)
- Certificate Validation

Temporal Token Validation

The validation of Temporal Tokens is requested by third parties / applications that have extracted them from their associated Time Stamp Token. This validation is performed by the Element Manager.

There is no tight system integration between the application, Time Stamp Authority and Trusted Time Product. Therefore, Temporal Token validation requests (TTValReq) and responses are labor intensive and not fully automated.

The first supported transport mechanism for Temporal Token validation is HTTPS (HTTP and SSL). The NTMS web server is set up to handle secure, web-based requests for verification from non-NTMS user clients. The server application accesses the NTMS database and performs the verification.

When a TTValReq is received, the Element Manager validates the timing chain that produced that Temporal Token from the TSS to the NMIServer, or if none, the Stratum 1 TMC. This validation also includes validation that the associated Trusted Time Product certificates where not revoked when the Temporal Token was produced.

Once the validity or invalidity of the Temporal Token is determined, the Element Manager returns a digitally signed message indicting this status to the requester.

The exact nature of the Datum Temporal Token validation application is still to be determined (TBD).

Registration Authority (RA)

Registration Authority functionality is built into the NTMS to facilitate interoperability with a variety of third party Certification Authority products and services. Since each third party Certification Authority is uniquely implemented (even if using the standard IETF PKIX protocols), the Trusted Time Product which interfaces with the Certification Authority has been modified to support it.

To limit the scope of these modifications, the NTMS contains Registration Authority functionality and translates between TSS/TMC certificate messages and third party CA/RA message formats. This approach minimizes the TSS and TMC software alterations when using different CA product/service vendors.

The NTMS RA functionality is created using certificate tools from Datum's initial partner (for example, Entrust). These tools enable the NTMS to create proper certificate management messages for processing by the initial partner's Certification Authority. Ideally, to minimize NTMS changes, CA partners are selected based upon their support of the IETF PKIX protocols—specifically, RFC 2459, RFC 2510, RFC 2511, and the OCSP Internet Draft.

NOTE: The Element Manager is the user interface to the Registration Authority. When the NTMS RA functionality is invoked by the EM, the RA creates a Certificate Request Message (it is compliant with RFC 2511). The request contains the Trusted Time Product name, TTDS Stratum Level, TTP serial number, and TTDS Policy as specified by the NTMS Element Manager.

This certificate request is then sent to the Certification Authority via a secure link. It is expected that this secure link initially uses the TLS/SSL protocol, but this depends upon the CA product or service. The CA returns a TTDS Certificate which the RA forwards on to the Trusted Time Product using SNMP.

The RA archives all CA-issued Trusted Time Product certificates to the NTMS database. It also produces CD-ROMs containing the certificates of the Stratum 1 TMCs and the TTDS CA Root (CERTTTDS[CATTD]) for export to the NMIServer for use in time certifying the Stratum 1 TMCs.

The RA operator (that is, Security Officer) performs telephonic verification of the certificates with the NMIServer operators. For this, special software on the RA needs to display the certificate fields along with the first 32 hex characters of the SHA-1 hashes of the subject public key and CA digital signature fields.

The SO and the NMIServer operators telephonically verify these fields and partial hashes to ascertain the validity of the exchanged certificates.

The RA acts as a gatekeeper to ensure that only the properly authorized Trusted Time Products are certified for use in a particular Trusted Time Distribution Service. All NTMS RA functions require the presence of the Security Officer.

Certificate Validation

The NTMS implements an OCSP Responder (OCSPR) in order to provide timely certificate revocation information distribution. The OCSPR is a standalone server which handles OCSP certificate status check requests from the TTI elements. The OCSPR contains a tamper-resistant hardware device for the generation of digital signatures. The operation of the OCSPR follows the most recent IETF PKIX OCSP Internet Draft.

NOTE: Because the current deployment of the Datum Trusted Time Product involves small quantities of units, it is easier to implement if a traditional certificate revocation list (CRL) is used instead of an OCSPR. The OCSPR functionality is best geared toward a TTDS operation. When the CA publishes a CRL, the NTMS must push it to all fielded units. Traditional CRL support must be phased out as the installed base of TTP increases.

Appendix A: Internet X.509 PKI Time Stamp Protocol

Internet X.509 Public Key Infrastructure Time Stamp Protocol (TSP)

Internet Draft
PKIX Working Group
expires in six months

C. Adams (Entrust Technologies)
P. Cain (BBN)
D. Pinkas (Bull)
R. Zuccherato (Entrust Technologies)

April 2000

Internet X.509 Public Key Infrastructure
Time Stamp Protocol (TSP)
<draft-ietf-pkix-time-stamp-07.txt>

Status of this Memo

This document is an Internet-Draft and is in full conformance with all provisions of Section 10 of RFC 2026.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html.

Abstract

A time stamping service allows to prove that a datum existed before a particular time and can be used by a Trusted Third Party (TTP) as one component in building reliable non-repudiation services (see [ISONR]). This document describes the format of a request sent to a Time Stamping Authority (TSA) and of the response that is returned. An example of how to prove that a digital signature was generated during the validity period of a public key certificate is given in an annex.

The key words "MUST", "MUST NOT", "REQUIRED", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document (in uppercase, as shown) are to be interpreted as described in [RFC2119].

1. Introduction

In order to associate a datum with a particular point in time, a Time Stamp Authority (TSA) may need to be used. This Trusted Third Party provides a "proof-of-existence" for this particular datum at an instant in time.

The TSA's role is to time stamp a datum to establish evidence indicating the time at which the datum existed. This can then be used, for example, to verify that a digital signature was applied to a message before the corresponding certificate was revoked thus allowing a revoked public key certificate to be used for verifying signatures created prior to the time of revocation. This is an important public key infrastructure operation. The TSA can also be used to indicate the time of submission when a deadline is critical, or to indicate the time of transaction for entries in a log. An exhaustive list of possible uses of a TSA is beyond the scope of this document.

2. The TSA

The TSA is a TTP that creates time stamp tokens in order to indicate that a datum existed at a particular point in time.

For the remainder of this document a "valid request" shall mean one that can be decoded correctly, is of the form specified in Section 2.4, and is from a supported TSA subscriber.

2.1. Requirements of the TSA

The TSA is REQUIRED:

1. to use a trustworthy source of time.

2. to include a trustworthy time value for each time stamp token.

3. to include a monotonically incrementing integer for each newly generated time stamp token.

4. to produce a time stamp token upon receiving a valid request from the requester, when it is possible.

5. to include within each time stamp token an identifier to uniquely indicate the security policy under which the token was created.

6. to only time stamp a hash representation of the datum, i.e.

a data imprint associated with a one-way collision resistant hash-function OID.

7. to examine the OID of the one-way collision resistant hash-function and to verify that the hash value length is consistent with the hash algorithm.

8. not to examine the imprint being time stamped in any way (other than to check its length, as specified in the previous bullet).

9. not to include any identification of the requesting entity in the time stamp tokens.

10. to sign each time stamp token using a key generated exclusively for this purpose and have this property of the key indicated on the corresponding certificate.

11. to include additional information in the time stamp token, if asked by the requester using the extensions field, only for the extensions that are supported by the TSA. If this is not possible, the TSA SHALL respond with an error message.

2.2. TSA Transactions

As the first message of this mechanism, the requesting entity requests a time stamp token by sending a request (which is or includes a TimeStampReq, as defined below) to the Time Stamping Authority. As the second message, the Time Stamping Authority responds by sending a response (which is or includes a TimeStampResp, as defined below) to the requesting entity.

Upon receiving the response (which is or includes a TimeStampResp, as defined below), the requesting entity SHALL verify the status error returned in the response and if no error is present it SHALL verify the various fields contained in the TimeStampToken and the validity of the digital signature of the TimeStampToken. In particular, it SHALL verify that what was time stamped corresponds to what was requested to be time stamped. The requester SHALL verify that the TimeStampToken contains the correct certificate identifier of the TSA, the correct data imprint and the correct hash algorithm OID. It SHALL then verify the timeliness of the response by verifying either the time included in the response against a local trusted time reference, if one is available, and/or the value of the nonce (large random number with a high probability that it is generated by the client only once) included in the response against the value included in the request. If any of the verifications above fails, the TimeStampToken SHALL be rejected.

Then, since the TSA's certificate may have been revoked, the status of the certificate SHOULD be checked (e.g. by checking the appropriate

CRL) to verify that the certificate is still valid.

Then, the client application SHOULD check the policy field to determine whether or not the policy under which the token was issued is acceptable for the application. The client MAY ignore this field if that is acceptable for the intended application.

2.3. Identification of the TSA

The TSA MUST sign all time stamp messages with one or more keys reserved specifically for that purpose. The corresponding certificate MUST contain only one instance of the extended key usage field extension as defined in [RFC2459] Section 4.2.1.13 with KeyPurposeID having value id-kp-timeStamping. This extension MUST be critical.

The following object identifier identifies the KeyPurposeID having value id-kp-timeStamping.


```
id-kp-timeStamping OBJECT IDENTIFIER ::= {iso(1)
                     identified-organization(3) dod(6)
                     internet(1) security(5) mechanisms(5) pkix(7)
                     kp (3) timestamping (3)}
```

A TSA's certificate MAY contain an Authority Information Access extension [RFC2459] in order to convey the method of contacting the TSA. The accessMethod field in this extension MUST contain the OID id-ad-timestamping:

The following object identifier identifies the access descriptors for timeStamping.

```
id-ad-timeStamping OBJECT IDENTIFIER ::= {iso(1)
                     identified-organization(3) dod(6)
                     internet(1) security(5) mechanisms(5) pkix(7)
                     ad (48) timestamping (3)}
```

The value of the accessLocation field defines the transport (e.g. HTTP) used to access the TSA and may contain other transport dependent information(e.g. a URL).

2.4. Request and Response Formats

2.4.1. Request Format

A time stamping request is as follows:

```
TimeStampReq ::= SEQUENCE  {
     version                      Integer  { v1(1) },
     messageImprint               MessageImprint,
       --a hash algorithm OID and the hash value of the data to be
       --time stamped
```

```
     reqPolicy                PolicyInformation      OPTIONAL,
     nonce                    Integer                OPTIONAL,
     certReq                  BOOLEAN                DEFAULT FALSE,
     extensions               [0] Extensions         OPTIONAL
}
```

The version field (currently v1) describes the version of the time stamp request.

The messageImprint field SHALL contain the hash of the datum to be time stamped. The hash is represented as an OCTET STRING. Its length MUST match the length of the hash value for that algorithm (e.g. 20 bytes for SHA-1 or 16 bytes for MD5).

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

The hash algorithm indicated in the hashAlgorithm field MUST be a known hash algorithm (one-way and collision resistant).

The reqPolicy field, if included, indicates the policy under which the TimeStampToken should be provided. PolicyInformation is defined in Section 4.2.1.5 of [RFC2459].

The nonce, if included, allows to verify the timeliness of the response when no local clock is available. The nonce is a large random number with a high probability that the client generates it only once (e.g. a 64 bit integer). In such a case the same nonce value shall be included in the response, otherwise the response shall be rejected.

If the certReq field is present and set to true, the TSA’s public key certificate that is referenced by the ESSCertID attribute must be provided by the TSA in the Certificate Values attribute and incorporated in the response. The Certificate Values attribute may also contain other certificates.

If the certReq field is missing, or if the certReq field is present and set to false then no Certificate Values attribute shall be present.

The extensions field is a generic way to add additional information to the request in the future. Extensions is defined in [RFC 2459]. If an extension, whether it is marked critical or not critical, is used by a requester but is not recognized by a time stamping server, the server SHALL not issue a token and SHALL return a failure (unacceptedExtension).

The time stamp request does not identify the requester, as this

information is not validated by the TSA (See Section 2.1).

In situations where the TSA requires the identity of the requesting entity, alternate identification /authentication means have to be used (e.g. CMS encapsulation [CMS] or TLS authentication [RFC2246]).

2.4.2. Response Format

A time stamping response is as follows:

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

The status is based on the definition of status in section 3.2.3 of [RFC2510] as follows:

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

When the Status contains the value zero a Time Stamp Token is present.


```
PKIStatus ::= INTEGER {
         granted                 (0),
         -- When the Status contains the value zero a Time Stamp Token
         -- is present.
         grantedWithMods         (1),
         rejection               (2),
         waiting                 (3),
         revocationWarning       (4),
         revocationNotification (5),
         keyUpdateWarning        (6)
         -- Otherwise, the Time Stamp Token is not present
     }
```

When the Time Stamp Token is not present the failInfo indicates the reason why the time stamp request was rejected and may be one of the following values.

```
PKIFailureInfo ::= BIT STRING {
    badAlg               (0),
      -- unrecognized or unsupported Algorithm Identifier
    badRequest           (2),
      -- transaction not permitted or supported
    badDataFormat        (5),
      -- the data submitted has the wrong format
```

```
    timeNotAvailable    (14),
      -- the TSA's time source is not available
    unacceptedPolicy    (15),
      -- the requested TSA policy is not supported by the TSA.
    unacceptedExtension (16),
      -- the requested extension is not supported by the TSA.
    addInfoNotAvailable (17)
      -- the additional information requested could not be understood
      -- or is not available
     }
```

These are the only values of PKIFailureInfo that are supported.
Compliant servers MUST NOT produce any other values.
Compliant clients MAY ignore any other values.

The statusString field of PKIStatusInfo MAY be used to include reason text such as "messageImprint field is not correctly formatted".

If the error code returned is different from zero, then the TimeStampToken is not returned.

A TimeStampToken is as follows. It is defined as a ContentInfo ([CMS]) and SHALL encapsulate a signed data content type.

```
TimeStampToken ::= ContentInfo
  -- contentType is id-signedData ([CMS])
  -- content is SignedData ([CMS])
```

The fields of type EncapsulatedContentInfo of the SignedData construct have the following meanings:

eContentType is an object identifier that uniquely specifies the content type. For a time stamping token it is defined as:

```
id-smime-ct-TSTInfo  OBJECT IDENTIFIER ::= {id-smime-ct 4}
```

with:

```
id-smime-ct           OBJECT IDENTIFIER ::= { id-smime 1 }
id-smime              OBJECT IDENTIFIER ::= { iso(1) member-body(2)
                      us(840) rsadsi(113549) pkcs(1) pkcs-9(9) 16 }
```

eContent is the content itself, carried as an octet string.
The eContent SHALL be the DER-encoded value of TSTInfo.

The time stamp token MUST NOT contain any signatures other than the signature of the TSA. The certificate identifier (ESSCertID) of the TSA certificate shall be included as a signerInfo attribute.

```
TSTInfo ::= SEQUENCE  {
     version                     Integer  { v1(1) },
```

```
     policy                  PolicyInformation,
     messageImprint          MessageImprint,
       -- MUST have the same value as the similar field in
       -- TimeStampReq
     serialNumber            Integer,
     genTime                 GeneralizedTime,
     accuracy                Accuracy            OPTIONAL,
     nonce                   Integer             OPTIONAL,
       -- MUST be present if the similar field was present
       -- in TimeStampReq. In that case it must have the same value.
     tsa                     GeneralName         OPTIONAL,

     extensions              [0] Extensions      OPTIONAL
}

The version field (currently v1) describes the version of the
time stamp token.

    Conforming timestamping servers MUST be able to provide version 1
    time stamp tokens. Among the optional fields, only the nonce field
    MUST be supported.

    Conforming timestamping requesters MUST be able to recognize
    version 1 time stamp tokens with all the optional fields present,
    but are not mandated to understand the semantics of any extension,
    if present.

The policy field MUST indicate the TSA's policy under which the response
was produced. If a similar field was present in the TimeStampReq, then
it MUST have the same value, otherwise an error (badRequest) MUST be
returned. This policy MAY include the following types of information
(although this list is certainly not exhaustive):

* The conditions under which the time stamp may be used.

Adams, Cain, Pinkas, Zuccherato                          [ Page 7 ]

Time Stamp Protocol                Document Expiration : October 2000

* The availability of a time-stamp log, to allow later verification
  that a time-stamp token is authentic.

The messageImprint must have the same value as the similar field in
TimeStampReq, provided that the size of the hash value matches the
expected size of the hash algorithm identified in hashAlgorithm.

The serialNumber field shall include a strictly monotonically
increasing integer from one TimeStampToken to the next (e.g. 45, 236,
245, 1023, ...). This guarantees that each token is unique and allows
to compare the ordering of two time stamps from the same TSA. This is
useful in particular when two time stamps from the same TSA bear the
same time. This field also provides the way to build a unique
identifier to reference the token. It should be noticed that the
monotonic property must remain valid even after a possible
interruption (e.g. crash) of the service.
```

genTime is the time at which the time stamp has been created by the TSA. The ASN.1 GeneralizedTime syntax can include fraction-of-second details. Such syntax, without the restrictions from [RFC 2459] Section 4.1.2.5.2, where GeneralizedTime is limited to represent time with one second, may to be used here. However, when there is no need to have a precision better than the second, then GeneralizedTime with a precision limited to one second should be used (as in [RFC 2459]).

The syntax is: YYYYMMDDhhmmss[.s...]Z
Example: 19990609001326.34352Z

X.690 | ISO/IEC 8825-1 provides the restrictions for a DER-encoding.

The encoding shall terminate with a "Z". The decimal point element, if present, shall be the point option ".". The fractional-seconds elements, if present, shall omit all trailing 0's; if the elements correspond to 0, they shall be wholly omitted, and the decimal point element also shall be omitted.

Midnight (GMT) shall be represented in the form: "YYYYMMDD000000Z" where "YYYYMMDD" represents the day following the midnight in question. Here are a few examples of valid representations:

```
    "19920521000000Z"
    "19920622123421Z"
    "19920722132100.3Z"
```

accuracy represents the time deviation around the UTC time contained in GeneralizedTime.

```
Accuracy ::= SEQUENCE {
                seconds    [1] INTEGER              OPTIONAL,
                millis     [2] INTEGER  (1..999)    OPTIONAL,
                micros     [3] INTEGER  (1..999)    OPTIONAL
}
```

By adding the accuracy value to the GeneralizedTime, an upper limit of the time at which the time stamp has been created by the TSA can

be obtained. In the same way, by subtracting the accuracy to the GeneralizedTime, a lower limit of the time at which the time stamp has been created by the TSA can be obtained.

accuracy can be decomposed in seconds, milliseconds (between 1-999) and microseconds (1-999), all expressed as integer.

When the accuracy optional field is not present, then the accuracy may be available through other means, e.g. the PolicyInformation.

The nonce field MUST be present if it was present in the TimeStampReq. In such a case it MUST equal the value provided in the TimeStampReq

structure.

The purpose of the tsa field is to give an hint in identifying the name of the TSA. If present, it MUST correspond to one of the subject names included in the certificate that is to be used to verify the token. However, the actual identification of the entity that signed the response will always occur through the use of the certificate identifier (ESSCertID Attribute) which is part of the signerInfo (See Section 5 of [ESS]).

extensions is a generic way to add additional information in the future. Extensions is defined in [RFC 2459].

Particular extension field types may be specified in standards or may be defined and registered by any organization or community.

3. Transports

There is no mandatory transport mechanism for TSA messages in this document. The mechanisms described below are optional; additional optional mechanisms (e.g., based on TCP or HTTP) may be defined in the future.

3.1. Time Stamp Protocol Using E-mail

This section specifies a means for conveying ASN.1-encoded messages for the protocol exchanges described in Section 2 and Appendix D via Internet mail.

A simple MIME object is specified as follows:

```
Content-Type: application/time stamp
Content-Transfer-Encoding: base64

<<the ASN.1 DER-encoded Time Stamp message, base64-encoded>>
```

This MIME object can be sent and received using common MIME processing engines and provides a simple Internet mail transport for Time Stamp messages.

3.2. File Based Protocol

A file containing a time stamp message MUST contain only the DER encoding of one TSA message, i.e. there MUST be no extraneous header or trailer information in the file. Such files can be used to transport time stamp messages using for example, FTP.

3.3. Socket Based Protocol

The following simple TCP-based protocol is to be used for transport of TSA messages. This protocol is suitable for cases where an entity initiates a transaction and can poll to pick up the results.

The protocol basically assumes a listener process on a TSA that can accept TSA messages on a well-defined port (IP port number 318).

Typically an initiator binds to this port and submits the initial TSA message. The responder replies with a TSA message and/or with a reference number to be used later when polling for the actual TSA message response.

If a number of TSA response messages are to be produced for a given request (say if a receipt must be sent before the actual token can be produced) then a new polling reference is also returned.

When the final TSA response message has been picked up by the initiator then no new polling reference is supplied.

The initiator of a transaction sends a "direct TCP-based TSA message" to the recipient. The recipient responds with a similar message.

A "direct TCP-based TSA message" consists of:
length (32-bits), flag (8-bits), value (defined below)

The length field contains the number of octets of the remainder of the message (i.e., number of octets of "value" plus one). All 32-bit values in this protocol are specified to be in network byte order.

```
    Message name   flag    value
    tsaMsg         '00'H   DER-encoded TSA message
      -- TSA message
    pollRep        '01'H   polling reference (32 bits),
                           time-to-check-back (32 bits)
      -- poll response where no TSA message response ready; use polling
      -- reference value (and estimated time value) for later polling
    pollReq        '02'H   polling reference (32 bits)
      -- request for a TSA message response to initial message
    negPollRep     '03'H   '00'H
      -- no further polling responses (i.e., transaction complete)
    partialMsgRep  '04'H   next polling reference (32 bits),
                           time-to-check-back (32 bits),
                           DER-encoded TSA message
      -- partial response (receipt) to initial message plus new polling
```



```
      -- reference (and estimated time value) to use to get next part of
      -- response
    finalMsgRep    '05'H   DER-encoded TSA message
      -- final (and possibly sole) response to initial message
```

```
    errorMsgRep    '06'H    human readable error message
      -- produced when an error is detected (e.g., a polling reference
      -- is received which doesn't exist or is finished with)
```

The sequence of messages that can occur is:

a) entity sends tsaMsg and receives one of pollRep, negPollRep, partialMsgRep or finalMsgRep in response.
b) end entity sends pollReq message and receives one of negPollRep, partialMsgRep,finalMsgRep or errorMsgRep in response.

The "time-to-check-back" parameter is a 32-bit integer, defined to be the number of seconds that have elapsed since midnight, January 1, 1970, coordinated universal time.

It provides an estimate of the time that the end entity should send its next pollReq.

3.4. Time Stamp Protocol via HTTP

This subsection specifies a means for conveying ASN.1-encoded messages for the protocol exchanges described in Section 2 and Appendix D via the HyperText Transfer Protocol.

A simple MIME object is specified as follows.

Content-Type: application/time stamp

```
   <<the ASN.1 DER-encoded Time Stamp message>>
```

This MIME object can be sent and received using common HTTP processing engines over WWW links and provides a simple browser-server transport for Time Stamp messages.

Upon receiving a valid request, the server MUST respond with either a valid response with content type application/time stamp or with an HTTP error.

4. Security Considerations

This entire document concerns security considerations.

When designing a TSA service, the following considerations have been identified that have an impact upon the validity or "trust" in the time stamp token.

1. When there is a reason to believe that the TSA can no longer be trusted but the TSA private key has not been compromised, the authority's certificate SHALL be revoked.

Thus, at any future time, the tokens signed with the corresponding key will not considered as valid.

2. When the TSA private key has been compromised, then the corresponding certificate SHALL be revoked. In this case, any token signed by the TSA using that private key cannot be trusted anymore. For this reason, it is imperative that the TSA's private key be guarded with proper security and controls in order to minimize the possibility of compromise. In case the private key does become compromised, an audit trail of all tokens generated by the TSA MAY provide a means to discriminate between genuine and false backdated tokens.

   A double time stamp from two different TSAs is another way to address this issue.

3. The TSA signing key MUST be of a sufficient length to allow for a sufficiently long lifetime. Even if this is done, the key will have a finite lifetime. Thus, any token signed by the TSA SHOULD be time stamped again (if authentic copies of old CRLs are available) or notarized (if they aren't) at a later date to renew the trust that exists in the TSA's signature. Time stamp tokens could also be kept with an Evidence Recording Authority to maintain this trust.

4. An application using the TSA service SHOULD be concerned about the amount of time it is willing to wait for a response. A 'man-in-the-middle' attack can introduce delays. Thus, any TimeStampResp that takes more than an acceptable period of time SHOULD be considered suspect.

5. If different entities obtain timestamps on the same data object using the same hash algorithm, or a single entity obtains multiple timestamps on the same object, the generated time stamp tokens will include identical message imprints; as a result, an observer with access to those time stamp tokens could infer that the timestamps may refer to the same underlying data.

5. Intellectual Property Rights

The IETF takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Information on the IETF's procedures with respect to rights in standards-track and standards-related documentation can be found in BCP-11. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementors or users of this specification can be obtained from the IETF Secretariat.

The IETF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights which may cover technology that may be required to practice this standard. Please address the information to the IETF Executive Director.

The following eight (8) United States Patents related to time stamping, listed in chronological order, are known by the authors to exist at this time. This may not be an exhaustive list. Other patents MAY exist or be issued at any time.

Implementers of this protocol SHOULD perform their own patent search and determine whether or not any encumbrances exist on their implementation.

Users of this protocol SHOULD perform their own patent search and determine whether or not any encumbrances exist on the use of this standard.

5,001,752Public/Key Date-Time Notary Facility
Filing date: October 13, 1989
Issued: March 19, 1991
Inventor: Addison M. Fischer

5,022,080Electronic Notary
Filing date: April 16, 1989
Issued: June 4, 1991
Inventors: Robert T. Durst, Kevin D. Hunter

5,136,643Public/Key Date-Time Notary Facility
Filing date: December 20, 1990
Issued: August 4, 1992
Inventor: Addison M. Fischer
Note: This is a continuation of patent 5,001,752.)

5,136,646Digital Document Time-Stamping with Catenate Certificate
Filing date: August 2, 1990
Issued: August 4, 1992
Inventors: Stuart A. Haber, Wakefield S. Stornetta Jr.
(assignee) Bell Communications Research, Inc.,

5,136,647Method for Secure Time-Stamping of Digital Documents
Filing date: August 2, 1990
Issued: August 4, 1992
Inventors: Stuart A. Haber, Wakefield S. Stornetta Jr.
(assignee) Bell Communications Research, Inc.,

5,373,561Method of Extending the Validity of a Cryptographic Certificate
Filing date: December 21, 1992
Issued: December 13, 1994

Inventors: Stuart A. Haber, Wakefield S. Stornetta Jr.
(assignee) Bell Communications Research, Inc.,

5,422,953 Personal Date/Time Notary Device
Filing date: May 5, 1993
Issued: June 6, 1995
Inventor: Addison M. Fischer

5,781,629Digital Document Authentication System
Filing date: February 21, 1997
Issued: July 14, 1998
Inventor: Stuart A. Haber, Wakefield S. Stornetta Jr.
(assignee) Surety Technologies, Inc.,

6. References

[RFC2510] C. Adams, S. Farrell, "Internet X.509 Public Key Infrastructure, Certificate Management Protocols," RFC 2510,March 1999.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC2246] T. Dierks, C. Allen, "The TLS Protocol, Version 1.0," RFC 2246, January 1999.

[ESS] P. Hoffman, "Enhanced Security Services for S/MIME", RFC 2634, June 1999.

[CMS] R. Housley, "Cryptographic Message Syntax", RFC 2630, June 1999.

[RFC2459] R. Housley, W. Ford, W. Polk, D. Solo, "Internet X.509 Public Key Infrastructure, Certificate and CRL Profile," RFC 2459, January 1999.

[ISONR] ISO/IEC 10181-5: Security Frameworks in Open Systems. Non-Repudiation Framework. April 1997.

7. Authors' Addresses

Carlisle Adams
Entrust Technologies
750 Heron Road
Ottawa, Ontario
K1V 1A7
CANADA
cadams@entrust.com

Pat Cain
BBN
70 Fawcett Street
Cambridge, MA 02138
U.S.A.
pcain@bbn.com

Denis Pinkas

Robert Zuccherato

Bull S.A.
12 rue de Paris
B.P. 59
78231 Le Pecq
FRANCE
Denis.Pinkas@bull.net

Entrust Technologies
750 Heron Road
Ottawa, Ontario
K1V 1A7
CANADA
robert.zuccherato@entrust.com

APPENDIX A - Signature time stamp attribute using CMS

One of the major use of time stamping is to time stamp a digital signature to prove that the digital signature was created before a given time. Should the corresponding public key certificate be revoked this allows to know whether the signature was created before or after the revocation date.

A sensible place to store a time stamp is in a [CMS] structure as an unsigned attribute.

This appendix defines a Signature time stamp attribute that may be used to time stamp a digital signature.

The following object identifier identifies the Signature time stamp attribute:

id-signatureTimeStampToken OBJECT IDENTIFIER ::= { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) id-aa (2) id-aa-timeStampToken (14)}

The Signature time stamp attribute value has ASN.1 type SignatureTimeStampToken:

SignatureTimeStampToken ::= TimeStampToken

The value of messageImprint field within TimeStampToken shall be a hash of the value of signature field within SignerInfo for the signedData being timestamped.

APPENDIX B - Placing a Signature At a Particular Point in Time

We present an example of a possible use of this general time stamping service. It places a signature at a particular point in time, from which the appropriate certificate status information (e.g. CRLs) MUST be checked. This application is intended to be used in conjunction with evidence generated using a digital signature mechanism.

Signatures can only be verified according to a non-repudiation policy. This policy MAY be implicit or explicit (i.e., indicated in the evidence provided by the signer). The non-repudiation policy can specify, among other things, the time period allowed by a signer to declare the compromise of a signature key used for the generation of digital signatures. Thus a signature may not be guaranteed to be valid until the termination of this time period.

To verify a digital signature, the following basic technique may be used:

A) Time stamping information needs to be obtained soon after the signature has been produced (e.g. within a few minutes or hours).

   1) The signature is presented to the Time Stamping Authority (TSA). The TSA then returns a TimeStampToken (TST) upon that signature.

   2) The invoker of the service must then verify that the TimeStampToken is correct.

B) The validity of the digital signature may then be verified in the following way:

   1) the Time stamp itself must be verified and it must be verified that it applies to the signature of the signer.

   2) The date/time indicated by the TSA in the Time Stamping Token must be retrieved.

   3) The certificate used by the signer must be identified

and retrieved.

4) The date/time indicated by the TSA must be inside the validity period of the signer's certificate.

5) The revocation information about that certificate, at the date/time of the Time Stamping operation, must be retrieved.

6) Should the certificate be revoked, then the date/time of revocation shall be later than the date/time indicated by the TSA

If all these conditions are successful, then the digital signature shall be declared as valid.

APPENDIX C MIME Registration

To: ietf-types@iana.org
Subject: Registration of MIME media type application/time stamp

MIME media type name: application

MIME subtype name: time stamp

Required parameters: None

Optional parameters: None

Encoding considerations: binary or Base64

Security considerations: Carries a request for a time stamp and the response. The response will be cryptographically signed.

Interoperability considerations: None

Published specification: IETF PKIX Working Group Draft on Time Stamp Protocols

Applications which use this media type: Time Stamp clients

Additional information:

Magic number(s): None
File extension(s): .TSA
Macintosh File Type Code(s): none

Person & email address to contact for further information:
Robert Zuccherato <robert.zuccherato@entrust.com>

```
Intended usage: COMMON

Author/Change controller:
Robert Zuccherato <robert.zuccherato@entrust.com>
```


Appendix D: ASN.1 Module using 1988 Syntax

```
  PKIXTSP {iso(1) identified-organization(3) dod(6) internet(1)
     security(5) mechanisms(5) pkix(7) id-mod(0) id-mod-tsp(13)}

  DEFINITIONS IMPLICIT TAGS ::=

  BEGIN

  -- EXPORTS ALL --

  IMPORTS

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

        GeneralName, PolicyInformation
        FROM PKIX1Implicit88 {iso(1) identified-organization(3)
        dod(6) internet(1) security(5) mechanisms(5) pkix(7)
        id-mod(0) id-pkix1-implicit-88(2)}

        ContentInfo FROM CryptographicMessageSyntax {iso(1)
        member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs-9(9)
        smime(16) modules(0) cms(1)} ;

                        --  Locally defined OIDs  --

-- Authority Information Access for TSA
```

```
id-pkix-ad-timestamping  OBJECT IDENTIFIER ::= {id-pkix-ad 3}
id-pkix-ad               OBJECT IDENTIFIER ::= {id-pkix 48}
id-pkix                  OBJECT IDENTIFIER ::= {iso(1)
                         identified-organization(3) dod(6)
                         internet(1) security(5) mechanisms(5) pkix(7)}

-- eContentType for a time stamping token

id-smime-ct-TSTInfo  OBJECT IDENTIFIER ::= { id-smime-ct 4 }
id-smime-ct          OBJECT IDENTIFIER ::= { id-smime 1 }
id-smime             OBJECT IDENTIFIER ::= { iso(1) member-body(2)
                     us(840) rsadsi(113549) pkcs(1) pkcs-9(9) 16 }

-- 2.4.1

TimeStampReq ::= SEQUENCE  {
     version                      INTEGER  { v1(1) },
     messageImprint               MessageImprint,
       --a hash algorithm OID and the hash value of the data to be
       --time stamped
     reqPolicy               PolicyInformation      OPTIONAL,
     nonce                   INTEGER                OPTIONAL,
     certReq                 BOOLEAN                DEFAULT FALSE,
```



```
extensions             [0] Extensions         OPTIONAL
}

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

-- 2.4.2


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

    -- The status is based on the definition of status
    -- in section 3.2.3 of [RFC2510]


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

PKIStatus ::= INTEGER {
         granted                (0),
         -- When the Status contains the value zero a Time Stamp Token
```

```
        -- is present.
        grantedWithMods        (1),
        rejection              (2),
        waiting                (3),
        revocationWarning      (4),
        revocationNotification (5),
        keyUpdateWarning       (6)
        -- Otherwise, the Time Stamp Token is not present
     }

    -- When the Time Stamp Token is not present
    -- failInfo indicates the reason why the
    -- time stamp request was rejected and
    -- may be one of the following values.

PKIFailureInfo ::= BIT STRING {
    badAlg               (0),
      -- unrecognized or unsupported Algorithm Identifier
    badRequest           (2),
      -- transaction not permitted or supported
    badDataFormat        (5),
      -- the data submitted has the wrong format
    timeNotAvailable    (14),
      -- the TSA's time source is not available
    unacceptedPolicy    (15),
      -- the requested TSA policy is not supported by the TSA.
```



```
    unacceptedExtension (16),
      -- the requested extension is not supported by the TSA.
    addInfoNotAvailable (17)
      -- the additional information requested could not be understood
      -- or is not available
     }


TimeStampToken ::= ContentInfo
     -- contentType is id-signedData as defined in [CMS]
     -- content is SignedData as defined in([CMS])
     -- eContentType within SignedData is id-ct-TSTInfo
     -- eContent within SignedData is TSTInfo

TSTInfo ::= SEQUENCE  {
     version                      INTEGER  { v1(1) },
     policy                       PolicyInformation,
     messageImprint               MessageImprint,
       -- MUST have the same value as the similar field in
       -- TimeStampReq
     serialNumber                 INTEGER,
     genTime                      GeneralizedTime,
     accuracy                     Accuracy            OPTIONAL,
```

```
     nonce                       INTEGER             OPTIONAL,
       -- MUST be present if the similar field was present
       -- in TimeStampReq. In that case it must have the same value.
     tsa                         GeneralName         OPTIONAL,
     extensions                  [0] Extensions      OPTIONAL
}

Accuracy ::= SEQUENCE {
                seconds    [1] INTEGER           OPTIONAL,
                millis     [2] INTEGER  (1..999) OPTIONAL,
                micros     [3] INTEGER  (1..999) OPTIONAL
}

END
```


APPENDIX E - Full Copyright Statement

Appendix B: Trusted Time Infrastructure

Introduction

The ability to time stamp e-documents is a fundamental business requirement. Critical to time stamping is the level of trust required in the actual source of time itself. How can a business be certain that the time contained in a time stamp is accurate? If multiple partners are conducting business transactions requiring time stamps, whose time will they agree to use? It is a fact that a server's built-in local clock can't be trusted since it can be adjusted easily to any value. Few recognize that a commonly used time reference, GPS, can be spoofed to indicate a different time. Most time stamp solutions today ignore both the issue of time source trust and the universal nature of time.

To solve these issues, Datum eBusiness Solutions has developed a range of Trusted Time™ Products capable of securely and verifiably distributing time from a country's legal time source down to the local time stamp applications themselves. Datum Trusted Time is time that is certified and traceable to a specific legal time source – a country's National Measurement Institute (NMI). In the United States, the National Institute of Standards and Technology (NIST) is legally established as the NMI. Time distributed by Datum's Trusted Time Products cannot be spoofed or altered.

This document provides a detailed technical description of the Datum Trusted Time Products and how they are used to create a Trusted Time Infrastructure to securely distribute time and issue IETF-compliant time stamps.

B.1 Coordinated Universal Time and Its Distribution

Coordinated Universal Time (UTC) is the world standard for time. The origin of UTC is the International Timing Authority (ITA) known as the International Bureau of Weights and Measures (BIPM) in France. National timing authorities in various countries discipline their atomic clocks to be within a few nanoseconds of that of BIPM's UTC. Each NMI then acts as a source of UTC for its country. By international agreement, the NMIs maintain audit records of their synchronization with BIPM UTC, thus providing verifiable sources of UTC within their countries. These NMIs supply time to hierarchies, or strata, of lower clocks that ultimately make UTC available to applications. This timing distribution hierarchy is summarized in Figure B-1. While the ITA-NMI timing relationship is documented and audited, the timing distribution hierarchy from the NMI down to the application has traditionally not been secure.

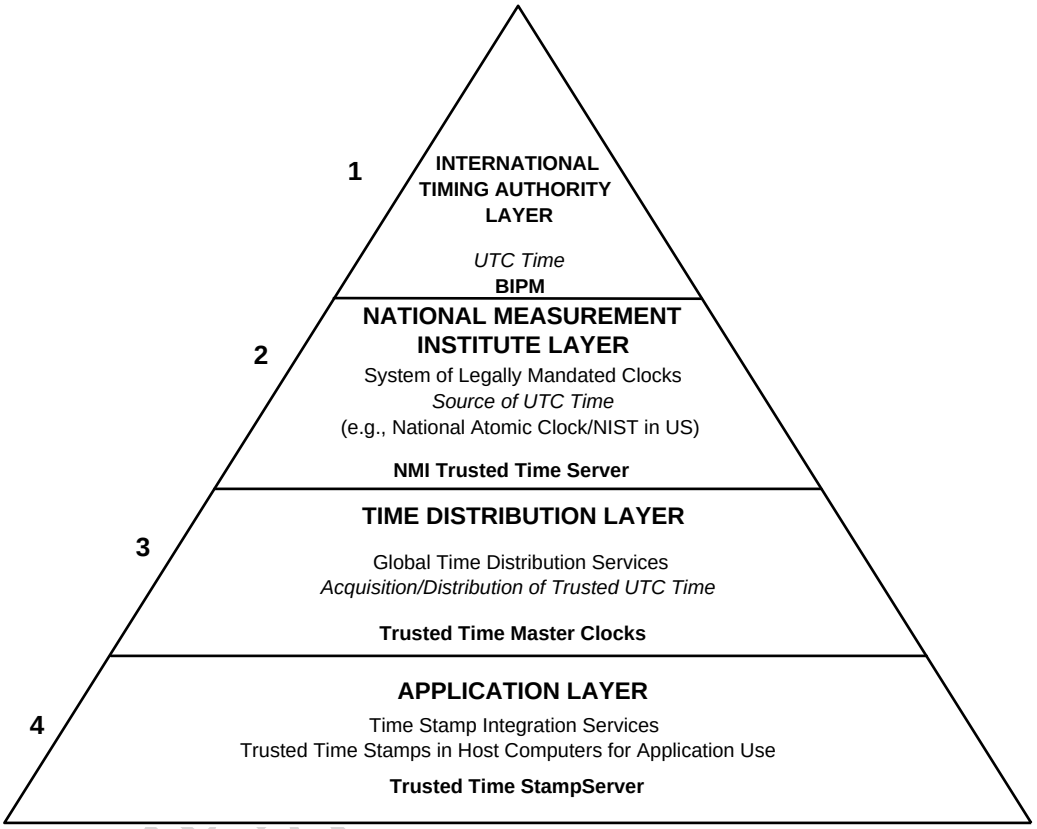

Figure B – 1 Global Timing Hierarchy

B.2 The Trusted Time Infrastructure

To protect and authenticate UTC distribution, Datum's Trusted Time Products (TTP) overlay onto the timing hierarchy, shown in Figure B-1, to create a Trusted Time Infrastructure (TTI) that provides the necessary security and audit mechanisms at each stratum level to enable the trusted distribution of UTC from the NMI to the application. The Datum products comprising the TTI consist of: the National Measurement Institute Server (NMIServer), the Trusted Master Clock (TMC), the TT StampServer (TSS) and the Network Time Management System (NTMS). The NMIServer, TMC and TSS are used to securely distribute time from the NMI to the local application. The NTMS is used to configure and manage the Trusted Time products. A detailed description of these products are located in section 5.0 of this appendix.

B.3 TTI Architecture

The general Trusted Time Infrastructure architecture is shown in Figure B-2. A TTI delivers trusted time from the NMI to the application layer, a time stamp server in this illustration.

Trusted time is created at the NMI using the NMIServer. The NMIServer is responsible for measuring the clock offsets of the TTI Stratum 1 TMC units and certifying them operational. The maximum allowable TMC offset from NMI UTC is 10mS. Currently, the offset measurement and time certification are performed via dial-up PSTN lines, but future NMIServer models will also use TCP/IP links. A Datum variant of the IETF draft protocol Secure NTP (S/NTP), called DS/NTP, is used to measure the clock offsets of the Stratum 1 TMCs. Datum is working with the IETF to align DS/NTP and S/NTP.

For redundancy, Datum recommends that two NMIServer units be located at the NMI, preferably at geographically separate locations. The NMI is responsible for monitoring both the accuracy of the NMIServer-produced time and the overall operation of the NMIServer. Note that a TTI can also be constructed in the event there is no NMI or where the NMI cannot host an NMIServer by simply using a TMC equipped with either a Rubidium or Cesium clock.

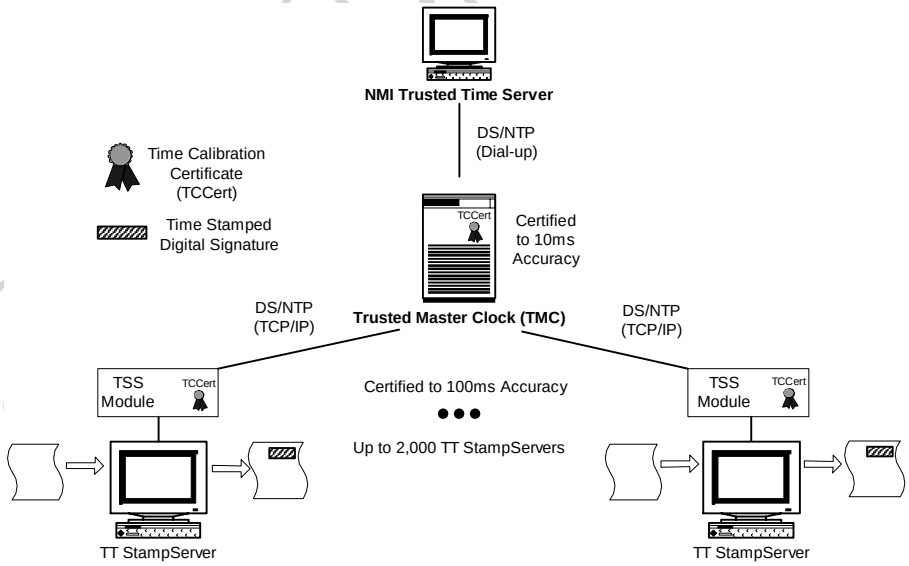

FIGURE B-2: TTI Architecture

The Stratum 1 layer of the TTI is comprised of TMCs. For redundancy, at least two TMC units should be used at this layer. The number of TMCs required at this stratum depends on the number of TSSs that must be managed. Up to 2,000 TSSs may be managed by two TMCs. Note that for very large TTI, another layer of

TMCs may be used to further distribute UTC to a very large number of TSSs. The NTMS assigns lower strata clocks to the Stratum 1 TMCs for which they are responsible for time certifying. The TMCs are responsible for determining the offset of the lower clocks and certifying them operational. The clock offsets of these lower TMCs and/or TSSs are measured by the higher level TMCs using DS/NTP over TCP/IP networks. The maximum allowable Stratum 2 TMC offset from NMI UTC is 20mS. The maximum allowable Stratum 2 TSS offset from NMI UTC is 100 mS. Using Datum patent pending technology, the TMC may also send small time corrections to the TSS in order to adjust the TSS clock to keep it within specification.

The TSSs operate as Time Stamp Authority (TSA) “engines” and are used by Time Stamp Authorities (TSAs) or other applications to implement time stamping as specified in the IETF PKIX Time Stamp draft standard. The TSSs create IETF Time Stamp Tokens (TST) containing trusted time, in the form of a Temporal Token (TT), embedded as an extension in the TST.

The TSA and other client applications access the TSSs using the Datum Trusted Time API Software Development Kit (TTAPI SDK). This SDK provides a rich, well-documented API, that provides straightforward access to all TSS functions. The TTAPI SDK documentation is available from Datum.

The Network Time Management System (NTMS) includes an element management system to remotely configure and monitor the NMIServer, TMC, and TSS equipment using SSLv3 as the secure transport protocol for the Datum Secure Management Protocol (DSMP) commands. In addition to the element manager, the NTMS includes a central Oracle database which is used to store all TMC and TSS certification logs and audit information. The NTMS also acts as the registration authority (RA) to the associated certification authority (CA) for the issuance of certificates to the TTP elements since it knows all TTP elements comprising the TTI.

For general monitoring purposes, the TTP will also support SNMPv1. TTI clients may use their SNMPv1 management equipment to monitor their TTI components, but they have no capability to remotely control them.

The certification authority (CA) referenced above is required to produce the device certificates for the TTP. These certificates are used for device authentication during protocol sessions so that a strong chain of trust is maintained for all Trusted Time Products in a TTI. The CA may be either a third party CA product/service or the NTMS’ internal CA. The NTMS’ internal CA is suitable for use in small TTI networks; however, large scale, high assurance TTI networks are expected to use third party CA products and services. The TTP units request their device certificates from the NTMS RA using the Datum Secure Management Protocol (DSMP).

The NTMS RA ensures that only authorized TTP elements receive certificates from the CA. The NTMS RA converts between the DSMP certificate management messages and the certificate message formats required by the CA. The NTMS is also responsible for distributing certificate revocation lists, published by the CA, to the TTP deployed in the TTI. This distribution will by done by FTP.

B.4 TTI Installation and Operation

This section describes the installation and operation of the Datum Trusted Time Products and how they are used to create a Trusted Time Infrastructure.

B.5 TTP Installation & Configuration

B.5.1 TTP Factory Certification

During the final stages of manufacture at Datum's factory, when the tamper shields are intact and operational, each NTMS, TMC, and TSS generates its DSA key pair. The resulting public DSA key and TTP serial number are sent to the Datum Factory CA where a Datum Factory Certificate is created for the TTP. The returned certificate is stored in the TTP's tamper protected memory. The DFCA root certificate is also returned to the TTP with its DFC. Once the DFC and DFCA root have been issued and successfully stored in the TTP units, the TTP units may be fielded.

B.5.2 NTMS Installation & Configuration

The NTMS must be installed in a physically secure location with strong physical access controls since the security of the NTMS and its database are important to maintaining the overall trust model of the TTI. In addition, the NTMS must be located behind a firewall for protection from external hackers.

During the installation process, the NTMS operators must be assigned their roles (e.g, Network Manager and Security Officer) and they must set their passwords. To commence operation, the NTMS must first acquire a TTI certificate from the TTI CA. The Security Officer must also load into the NTMS the serial numbers of the TTP units that it is expected to manage. Using the NTMS GUI, the Network Manager can construct a map of the overall TTI and can begin configuring remote TTP.

B.5.3 NMIServer Installation & Configuration

The NMIServer is located at the NMI. For high reliability networks, two NMIServer units should be installed at the NMI to provide redundant operation. For its timing source, the NMIServer will receive a 1 PPS, 10 MHz and major time input from the NMI. Optionally, an external Cesium reference may be used.

When the NMIServer is installed and powered-up, it will execute a series of self tests and it will verify that its tamper-protected memory is intact. If a tamper is detected, the NMIServer will enter an alarm mode that must be manually reset by the operator.

The NMIServer is initialized directly by the system operator using an attached keyboard and monitor. The login to the system is password controlled with a unique password assigned to each operator. Two distinct operator roles are provided: system administrator and security officer.

The system administrator assigns the PSTN numbers for the remote TMC units which the NMIServer is to certify. He also enters the Timing Service Fault Notification information, such as e-mail addresses, pager numbers and phone numbers, for notification of the Stratum 1 administrators in the event of a failure. The system administrator will also set the initial time in cases where automated time acquisition is not deployed via network, modem or hardwire (time code).

The NMIServer will operate either as a standalone unit or as an NTMS managed unit depending upon the NMI's security policy. For standalone operation, the security officer (SO) is responsible for loading the X.509 certificates of the Stratum 1 Trusted Master Clocks that it is responsible for time-certifying. When the NMIServer is managed by the NTMS, the certificate upload is performed by the NTMS.

B.5.4 TMC Installation & Configuration

When the TMC is installed and powered-up, it will execute a series of self tests and it will verify that its tamper-protected memory is intact. If a tamper is detected, the TMC enters an alarm mode that must be manually reset by the operator.

If the self tests are successfully passed, the TMC is initially configured by the local operator. Operator access is under password control with a unique password per operator with all operator actions logged. Local configuration consists of:

4. Assigning the TMC's local IP address; and
5. Entering the NTMS' IP addresses.

Once the TMC is locally configured, it is contacted by the NTMS. The Datum Secure Management Protocol (DSMP) is used to protect these management messages. As part of mutual authentication phase of DSMP, the NTMS and TMC exchange their Datum Factory Certificates (DFC), which were loaded into them during manufacture. The NTMS also verifies that the serial number of the TMC clock is one that it is expected to manage.

If the TMC is a legitimate unit, then the NTMS, acting as a PKI Registration Authority (RA), requests a TTI certificate for the TMC from the CA using either PKCS #10 or RFC 2511. In the Cert_Request, the IssuerName and SubjectName fields are fully specified by the NTMS operator. The certificate validity period (6 months to 2 years) and policy fields are set per the timing service's policy. The Key Usage Extension shall be set to digitalSignature and this extension must be flagged Critical.

The CA processes the Cert_Request and then returns the device certificate, the CA root certificate, and any associated certificate chain to the NTMS RA which then passes them to the TMC. The NTMS may also pass additional element configuration data such as the names or IP addresses of the TMC/TSS units which the TMC is responsible for time certifying. Once the TMC receives its TTI certificate and is properly configured for operation in the TTI, it is logged by the NTMS as fully installed and ready for time calibration. Once operational, should a TMC receive a request by a TTP with a different TTI certificate, it will reject that communication, log an alarm and report the event to its authorized NTMS.

In the case of a lost DFC (due to expiration, tamper or malfunction), the TMC operator must physically inspect the unit and manually reset the alarm condition(s). The unit then must be locally instructed to generate a key pair. The NTMS will establish an insecure link with the TMC and retrieve its new public key. The NTMS security operator and the local operator must verbally confirm via telephone that the key was properly received by the NTMS. The NTMS will then request a new certificate for the TMC from the CA and place the TMC back in service.

B.5.5 TSS Installation & Configuration

When the TSS is installed and powered-up, it will execute a series of self tests and it will verify that its tamper-protected memory is intact. If a tamper is detected, the TSS enters an alarm mode that must be manually reset by the operator.

If the self tests are successfully passed, the TSS is locally configured by the local operator. Operator access is under password control with a unique password per operator with all operator actions logged. Local configuration consists of:

1. Assigning the TSS's local IP address (manually entered); and
2. Entering the NTMS' IP addresses.

Once the TSS is locally configured, it is contacted by the NTMS via DSMP. As part of this protocol, the NTMS and TSS exchange their Datum Factory Certificates (DFC), which were loaded into them during manufacture in order to mutually authenticate each other. The NTMS also verifies that the serial number of the TSS clock is one that it is expected to manage.

If the TSS is a legitimate unit, then the NTMS, invoking its RA functionality, requests a TTI certificate for the TSS from the TTI CA using either PKCS #10 or RFC 2511. In the Cert_Request, the IssuerName and SubjectName fields are fully specified by the NTMS operator. The certificate validity period (6 months to 2 years) and policy fields are set per the timing service's policy. The Key Usage Extension shall be set to digitalSignature and this extension must be flagged Critical.

The CA processes the Cert_Request and returns the TTI device certificate, the CA root certificate, and any associated certificate chain to the NTMS RA which then passes them to the TSS. Once the TSS receives its TTI certificate, it is logged by the NTMS as fully installed and ready for time calibration. Once operational, should a TSS receive a request by a TTP with a different TTI certificate, it will reject that communication, log an alarm, and report the event to its authorized NTMS.

In the case of a lost DFC (due to expiration, tamper or malfunction), the operator must physically inspect the TSS and manually reset the alarm condition(s). The TSS then must be locally instructed to generate a key pair. The NTMS will then establish an insecure link with the TSS and retrieve its new public key. The NTMS security operator and the local operator must verbally confirm via telephone that the key was properly received. The NTMS will then request a new certificate for the TSS from the CA and place the TSS back in service.

During initialization, the TSS must also generate a time stamp key pair (TSS-TSA) and obtain a certificate for the TSS-TSA public key. This certificate may come from the CA associated with the Time Stamp Authority (TSA) or it may come from the TTI CA. If it is the former, then the certificate request must be sent via the TTAPI; if it is the latter, then the certificate request must be sent to the TTI CA via DSMP.

B.6 TTI Operation

This section describes the operation of the Trusted Time Infrastructure.

B.6.1 Clock Time Calibration

Before a Trusted Time Product can be placed in service, it must be time calibrated by a higher stratum level Trusted Time clock of the same TTI. Time calibration is performed using the Datum Secure NTP (DS/NTP) protocol which provides both mutual authentication of the upper and lower TTP units and a strongly authenticated measurement channel. At the conclusion of a successful time calibration cycle in which the lower clock is determined to be within tolerance, a Time Calibration Certificate (TCCert) is produced for that clock. If the lower clock is out of tolerance, the upper clock will send corrections to the lower clock and then the upper clock will perform a second time calibration cycle. If the lower clock is still out of calibration, the lower clock will be placed in a non-operational state. Operational Trusted Time Products in a TTI must

possess a valid TCCert before they can enter the operational mode. Further, TSSs cannot issue Time Stamp Tokens unless they possess a valid TCCert.

B.6.2 NMIServer to TMC Time Calibration

Every 24 hours, the NMIServer measures the clock offsets of the Stratum 1 TMCs using PSTN connections and the DS/NTP protocol. For Stratum 1 clocks within a 10 mSec offset UTC, the NMIServer returns a TCCert indicating the TMC is enabled for time calibration of lower clocks. The NMIServer TCCert Expiration Date is 72 hours from the time of TCCert issuance.

Once the TCCert has been issued and if the Stratum 1TMC's GPS is within the offset window, the TMC will update its internal clock time to that of the GPS'. If the GPS and the TMC internal clock times ever diverge from each other beyond the offset window, the TMC may issue a warning or place itself out of service depending on the TTI policy. For a high assurance network, the TTI policy may require the TMC to go offline in this event. For high availability systems, TTI policy may dictate that in the event of an unacceptable difference between the TMC and GPS, the TMC should immediately attempt to obtain another TCCert, and if the TMC is found to be within the allowable offset, to continue operation; otherwise, it should place itself offline. The fact that the Datum Trusted Time Products do not rely upon GPS or any other RF signal as their primary time source should not be overlooked. Since these RF signals can be locally jammed or spoofed they are unsuitable primary time sources for high assurance networks. is one of the critical security enhancements provided by Datum's Trusted Time architecture.

The NMIServer will log all clock offset measurement information to paper. The printed records will contain the following information:

1. UTC Time of Calibration;
2. Lower Clock Name;
3. Measured Offset;
4. NMIServer Certificate Serial Number; and
5. Lower Clock Certificate Serial Number

Should the NMIServer not be able to access a lower clock for 48 hours, it will log an alarm and activate an automatic e-mail message and pager alert to both NMI and the TTI operations personnel. These operators may also call each other by telephone for real time coordination of the fault resolution. Should the NMIServer receive a request to time certify an unknown Stratum 1 TMC, it will refuse the connection and log the error.

B.6.3 TMC to TMC Calibration

In certain TTI configurations, upper stratum TMC units may need to monitor the clock offset of lower stratum TMC units. In this case, each upper TMC will have a set of lower TMCs for which it is the Upper Trusted Master Clock; these sets of lower TMCs may be shared with another upper TMC. Using DS/NTP, each Upper TMC contacts each of the Lower TMCs in its set once every 24 hours and performs a time calibration measurement. If the Lower TMC's offset is within specification, the Upper TMC issues a TCCert to the Lower TMC. The allowable offset between Upper and Lower TMCs is 20 ms of UTC.

At the conclusion of a DS/NTP session and if the lower stratum TMC's GPS is within the offset window, that TMC will update its internal clock time to that of the GPS. If the GPS and the TMC internal clock times ever diverge from each other beyond the offset window, the TMC may place itself out of service depending on the TTI policy. For high availability systems, TTI policy may dictate that in the event of an unacceptable

difference between the TMC and GPS, the TMC should immediately attempt to obtain another TCCert, and if the TMC is found to still be within the allowable offset, to continue operation; otherwise, it should place itself offline.

When the NTMS informs the Upper TMC of the presence of a newly installed Lower TMC, the Upper TMC will immediately commence a TCCert calibration cycle. For ongoing operation, if an Upper TMC cannot contact a Lower TMC on the first attempt, it will retry every two hours. If after 48 hours, it still cannot contact a Lower TMC, it will issue an alarm to the NTMS that it has been unable to contact the TMC. The NTMS will then contact the TMC and perform diagnostics.

Should an Upper TMC receive a request to time certify either an equal or higher stratum level TMC or an unknown Lower TMC, it will refuse the connection and log the error. An alarm will also be logged if a lower TMC receives a request to time certify an upper TMC.

B.6.4 TMC to TSS Calibration

In a typical TTI, each TMC has a set of TSSs (TT StampServers) for which it is the Trusted Master Clock. This set of TSSs may be shared with another TMC. Using DS/NTP, each TMC contacts each of the TSSs in its set once per day and performs a time calibration measurement. If the TSS's time offset is within specification, the TMC issues a TCCert to the TSS. The allowable offset between the TMC and TSS is 100 ms of UTC.

If the TSS is equipped with an optional GPS system or other local time reference (CDMA, IRIG, WWVB, DCF77, etc.), once the TCCert has been issued and if the local GPS (or other time reference) is within the offset window, the TSS will update its internal clock time to that of the GPS. If the GPS and the TSS internal clock times ever diverge from each other beyond the offset window, the TSS may place itself out of service depending on the TTI policy. For a high assurance network, the TTI policy may be for the TSS to go offline in this event. For high availability systems, TTI policy may dictate that, in the event of an unacceptable difference between the TSS and GPS, the TSS should immediately attempt to obtain another TCCert, and if the TSS is found to still be within the allowable offset, to continue operation; otherwise, it should place itself offline.

When the NTMS informs a TMC of the presence of a newly installed TSS, the TMC will immediately commence a TCCert calibration cycle of the TSS. For ongoing operations, if a TMC cannot contact a TSS on the first attempt, it will retry every two hours. If after 48 hours, it still cannot contact a TSS, it will issue an alarm to the NTMS that it has been unable to contact the TSS. The NTMS will then contact the TSS host and perform diagnostics.

Should a TMC receive a request to time certify an equal or higher stratum level TSS or an unknown lower TSS, it will refuse the connection and log the error. An alarm should also be logged if the TSS receives a request to time certify a TMC.

B.6.5 Time Stamp Authority Operation

The TSS is designed to provide the core cryptographic and time functions of a Time Stamp Authority (TSA) as defined in the IETF PKIX Time Stamp draft. Message imprints to be time stamped are passed to the TSS via the Trusted Time API (TTAI). The TSS will create a Time Stamp Token (TST) for the message imprint with an embedded Temporal Token. The TST then is returned to the TSA via the TTAPI.

Note that for TST signatures, a second key pair is used by the TSS. When the TSS is initialized, either the NTMS or the TSA instructs the TSS to generate the TST signature public key pair. Once generated, the TSS must obtain a certificate for the public key. This certificate may be obtained either from the TSA's CA or the

TTI CA depending upon the TSA's policy. When the certificate is returned, the TSS validates the certificate and then stores it in non-volatile on-card memory.

Since no standards describe TSA design and since TSA features are application specific, all TSA operations specified/described in the PKIX document, except for the actual creation of the TST described above, must be implemented by the TSA itself. These operations include: end entity authentication; policy negotiation and determination; TST audit trail functions; billing; and certificate management with the TSA's CA.

B.6.6 TST Request and Creation

In order to obtain a Time Stamp Token, a client application issues a Time Stamp Request per the IETF PKIX Time Stamp Protocol Draft to the TSA. The TSA validates and parses the received request, and then it requests a TST from the TSS. To create the TST, the TSA passes the necessary data for embedding in the TST to the TSS. This data may consist of part or all of the following fields (defined in the IETF draft):

- Policy,
- Message Imprint,
- Serial Number,
- Time Accuracy,
- Nonce,
- TSA Name, and
- Extensions.

In response to the TSA request via the TTAPI, the TSS creates a Temporal Token (TT). It then generates a TST which contains the TSA-provided data and the TT. This TST is then returned to the TSA. The TSA then creates the IETF-specified Time Stamp Response, conveying the TST, back to the application.

B.6.7 TST Validation

Upon receipt of the TimeStampResp, the application follows the IETF draft and verifies the status error returned in the response, or if no error is present, it verifies the structure of the TST. The application verifies that the message imprint in the TST matches the original message imprint sent to the TSA and that the digital signature corresponds to the signed data in the TST. The application may also verify that the TSA and TSS certificates have not been revoked. It also verifies that, if a nonce had been sent to the TSA in the TimeStampReq, the nonce embedded in the received TST matches the original. If a specific policy action was requested of the TSA, the application ensures that it was executed.

In order to validate a TST at some future time, the client application will compose a TST Verification Request (TSTVerReq) which it sends to the TSA for action. The TSTVerReq contains at a minimum, the TST. The TSA will parse the TSTVerReq and will initiate a TTVerReq which it sends to the NTMS to validate the TT. The NTMS response to the TTVerReq will form one of the elements in the TSA's TSTVerResp to the requesting application. The TSA will also check the CRL for the TSA signature keys and the TST's consistency with the TSA policy in effect when that TST was issued.

Note that the formats of the TSTVerReq and TSTVerResp, the transport protocol between the application and the TSA, and the exact validation actions of the TSA are outside the scope of both this document and the IETF draft.

B.6.8 TT Validation

Temporal Token verification requests (TTVerReq) will be generated by a variety of third party applications. They may originate from the end user's application or from a TSA. All TTVerReq are directed to the NTMS for action.

When the NTMS receives a TTVerReq, it validates the TCCerts of the timing chain from the TSS to the NMIServer. This validation will also include validation of the TTP certificates to ensure that all devices where properly certified by the CA and that none of the TTP in the timing chain had their TTI certificates revoked or expired at the time the TT was issued. Once the validity or invalidity of the TT is determined, the NTMS will return a digitally signed message indicting this status to the requester (TTVerResp).

B.6.9 TTI Network Management

Central to the operation of the TTI is the Network Time Management System (NTMS). The NTMS securely configures and manages all the TTPs in a specific TTI. Indeed, a TTP cannot be placed in service without being configured by the TTI's NTMS. The TTP's configuration and status information is protected by the Datum Secure Management Protocol (DSMP) and is accessible only to the specific TTI' NTMS. DSMP provides mutual authentication of the communicating parties and it provides strong message integrity and authentication. The ability of the NTMS to securely remotely control the deployed TTP translates to a lower cost of operation of a TTI since the need to dispatch technicians is greatly reduced.

B.6.10 TTP Configuration

The configuration of the individual TTP elements by the NTMS is straightforward. It consists primarily of assigning the corresponding upper and lower TTP elements to which a particular TTP will interface. The NTMS performs this assignment by dragging and dropping the lower stratum TTP icons onto the higher stratum level TTP icon using a GUI map representation of the TTI.

B.6.11 TTP Diagnostics

Should a particular TTP enter an alarm state, the NTMS may command it to perform self-diagnostics. These diagnostics may be either on-line or off-line. When off-line diagnostics are being performed, the TTP is placed in an "out-of-service" state and actively notifies corresponding service-requesting entities that it is "out-of-service." TTP diagnostics include the following:

B.6.12 In-service tests:

- Non-service affecting cryptographic module tests
- CPU function tests
- Memory tests

B.6.13 Out-of-service tests:

- Memory tests (e.g., stuck bits)
- NIC loopback
- Cryptographic module tests

Should any of the tests fail, the TTP will be placed in the “out-of-service” mode which prevents it from being time-certified and from issuing time stamps. A failed unit may only be placed back in service by direct NTMS command or by local operator intervention coordinated with the NTMS.

B.6.14 Audit Trail

Each TTP keeps an audit log which tracks: local operator actions, NTMS-initiated actions, upper and lower TTP time certifications, and alarms. The TSS audit log also tracks the number of Time Stamp Tokens issued under each particular TCCert.

This audit trail is maintained in the TTP as a proprietary record format in its local database (or file) which is retrievable by the NTMS using DSMP. All audit records are time stamped by the TTP when they occur. Once it has received its TCCert and prior to issuing a TCCert or TST, the TTP shall record its configuration with the configuring operator’s ID in a time stamped audit record. When the NTMS retrieves the TTP audit trail records, it writes them to CDROM with a time stamp across the group of retrieved TTP records. Alarms and other audit trail records are detailed in the individual TTP design documents. The NTMS displays all retrieved TTP alarm and configuration change data for review by the NTMS network manager. This review by the network manager generates a specific audit record in the NTMS database. If needed, the network manager may use DSMP to instruct the TTP to execute a series of tests or he may correct a configuration error.

It should be noted that in the event of a tamper alarm, the TTP will attempt to contact the NTMS immediately to report this event. A tamper alarm may not be reset by remote command alone; local operator intervention is required.

The NTMS maintains audit trails for all its operators’ actions. Audit trails will also be kept for all NTMS management functions such as software download and TTP configuration. These records will be written to CDROM and time stamped as they occur.

B.6.15 Software Update

To reduce the ongoing cost of operations and to facilitate maintenance, the TTP are capable of having their software remotely updated by the NTMS via the Internet. Only the NMIServer may not provide this feature depending on the NMI’s security policies.

When Datum produces a new code release for a TTP, it signs the code using the DFCA root private key. Each NTMS can download the signed code, using FTP, from Datum. Each NTMS uses the DFCA root public key to verify the signature on the received code to ensure its authenticity. The NTMS then re-signs the new software using its private DSS key. It will also notify the TTP under its management that a new software release is available and what its version number is. The TTP will use FTP to download the new software release. Once the new software is downloaded, the TTP verifies the NTMS' signature. If it is valid, the code is accepted and made operational.

B.6.16 TTI X.509 Certificate Management

The use of an X.509 PKI is fundamental to the operation of the TTI. Each TTI Trusted Time Product and the NTMS use certificates to bind their public signature keys to their identities and to authorize them to provide various services. The Datum TTI System Architecture specifically enables the use of third party CA products and services in the TTI.

Each TTP obtains a TTI-specific certificate as follows. When a TTP is first installed, it uses DS/NTP to establish a secure connection with the NTMS. After verifying that it is a valid unit, the NTMS invokes its Registration Authority (RA) functionality to create a PKCS 10 Certificate Request Message (CRM). In the future, the NTMS will support RFC 2511-compliant CRM.

The CRM contains the TTP's name fields, public key (extracted from its Datum Factory Certificate), and the certificate lifetime (as specified by NTMS/TTI policy). The CRM is securely transported from the NTMS to the CA using the CA-specified protocol. The CA then verifies proper receipt of the CRM and, if correct, creates a TTI certificate for the TTP and returns it to the NTMS. The NTMS verifies that the certificate was properly created as it specified (i.e., the contents and signature are correct). If it is correct, the NTMS forwards the certificate on to the TTP using DSMP. The default TTP TTI certificate lifetime is 1 year.

The TSS must obtain a certificate for its TSS-TSA time stamping public key from either the TSA's CA, via TTAPI, or the TTI CA, via DSMP. This certificate must contain the extended key usage field extension as defined in RFC 2459 Section 4.2.1.13 with KeyPurposeID having value id-kp-timeStamping. This extension must be marked critical. The default TSS-TSA certificate lifetime is determined by TSA policy.

B.7 Summary

Datum is the first company to successfully address the issues of the creation and distribution of Trusted Time from a National Timing Authority. Leveraging its over 20 years of experience with atomic clocks and timing hierarchies, Datum has created a Trusted Time Infrastructure that overlays at every level of the traditional timing hierarchy to enable UTC from an NMI to be delivered in a completely trusted manner to the application. Since Datum's products do not rely on RF timing signals for their time, the Datum TTI is immune to traditional spoofing and jamming attacks. The remote management capabilities of Datum's products minimize cost of ownership and maximize reliability and availability.

The concept of a TTI will greatly enhance advanced e-business solutions and it will simplify contractual arrangements since UTC from NMIs can now be specified. This Datum infrastructure, combined with proven PKI technology, provides the strongest possible trust model to ensure the legal acceptance of the time associated with an electronic transaction.

For further information on Datum and its Trusted Time Products, please do not hesitate to contact Datum eBusiness Solutions at +1-781-372-3600.

B.8 Datum Trusted Time Products

B.8.1 Trusted Master Clock

The Trusted Master Clock (TMC) is a standalone secure server designed to perform time certifications of lower TMC and TSS devices. The TMC uses a version of the NT operating system that has been stripped of all non-essential features. The TMC housing is designed to meet FIPS 140-1 Level 3 physical protection and tamper detection requirements. Cryptographic calculations, including key generation, are performed by a dedicated hardware token that is validated to FIPS 140-1 L2. The private signing key of the TMC is never exported from the token. The token also contains a high quality random number generator which has been designed to meet FIPS 140-1 L3 RNG requirements for operation and test. Sensitive TMC cryptographic information is contained in non-volatile token memory.

A block diagram of the Trusted Master Clock is shown in Figure B-3. The TMC includes an NT PCI bus server, Chrysalis Luna 2 Token, a Rubidium oscillator (or external Cesium oscillator), a GPS receiver card, a custom Datum timing engine, a V.90 modem card, and an Ethernet 10/100BaseT NIC.

The Rubidium oscillator is the heart of the Trusted Master Clock. It provides an accurate and stable time base for the system. The GPS unit provides initial time synchronization, and serves as a time and frequency reference and as a system monitor for any equipment failures. The timing engine forms the main clock of the system. The modem card provides the dial-up connection to the National Measurement Institute, and the Ethernet port provides the network connection to the TT StampServers.

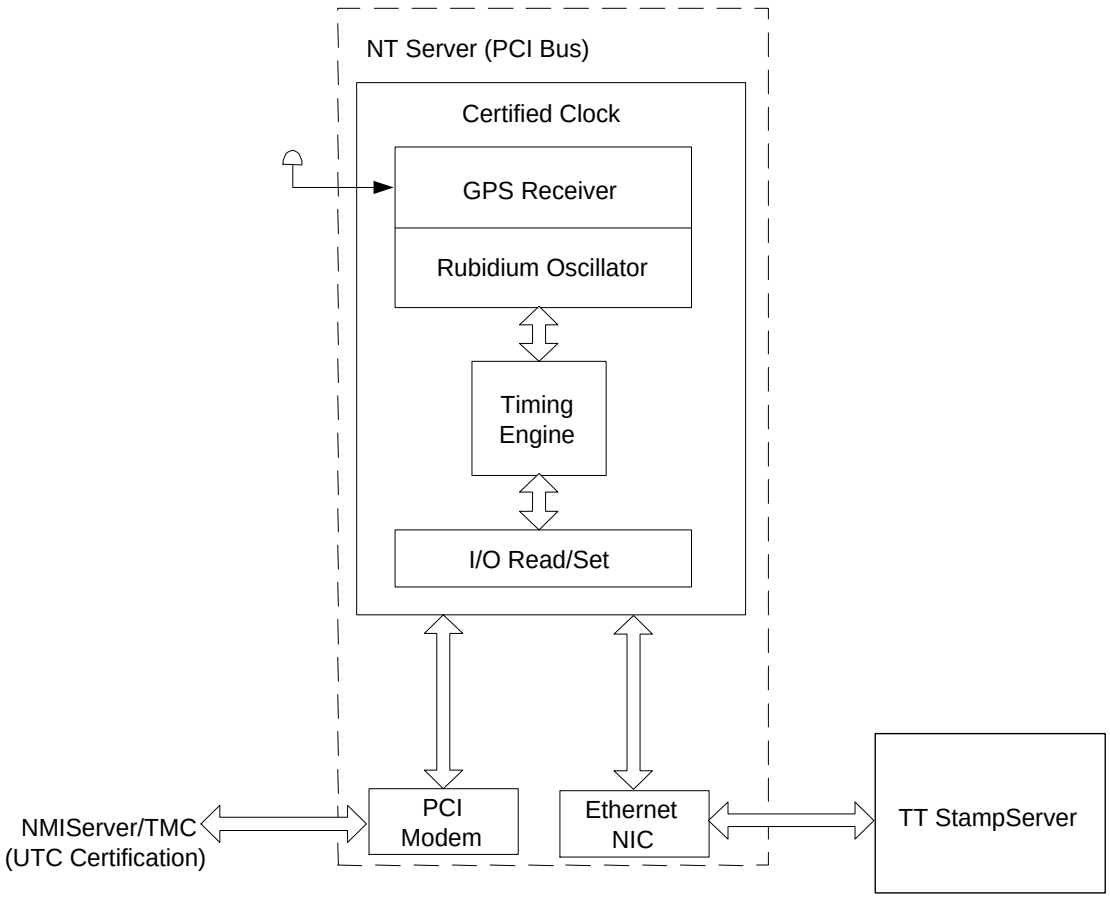

FIGURE B-3: Trusted Master Clock

The Trusted Master Clock generates a transaction log file, recording all exchanges with the National Measurement Institute and with the TT Stamp Servers. This log file records the time each transaction takes place, the measured clock offset, clock names (their unique identifiers), and the certificate serial numbers for each transaction.

In addition to periodic monitoring by the NMIServer and the NTMS, the Trusted Master Clock is designed to be self-monitoring. Any non-service-affecting failures – with no impact on the systems ability to deliver

National Measurement Institute-certified time to the StampServer – are reported to the NTMS immediately. All service-affecting failures will cause the system to take itself off-line immediately. This way, flawed time is never passed to lower stratum clocks.

B.8.2 NMI Trusted Time Server

The NMI Trusted Time Server (NMIServer) is a standalone secure server, based on the Trusted Master Clock, which is dedicated to the creation of trusted UTC time at the NMI. It uses a hardware-based FIPS 140-1 L2 certified cryptographic token for its cryptographic functions. This card also features a FIPS 140-1 L3 random number generator. The NMIServer is designed to meet FIPS 140-1 L3 requirements for physical security and tamper detection. Sensitive cryptographic information will be contained in tamper protected non-volatile memory. For additional security, its NT operating system is stripped of non-essential features.

External network connections supported by the NMIServer include 10/100BaseT network connections and V.90 modems. Note that all external TCP/IP connections must be protected by a firewall. The timing inputs to the NMIServer consist of a 1 PPS, 10 MHz and major time from the NMI or a Cesium reference.

The NMIServer maintains an audit trail of all operator actions, alarms, and time certifications performed. This audit trail is printed on a locally attached line printer. The NMIServer buffers up to 300 events in case the printer is offline. For additional security, the NMIServer is capable of e-mailing and paging specific NMIServer operators and TTI NTMS units in the event of a problem (e.g., “Tamper Alarm” or “Stratum 1 TMC Unreachable”).

If allowed by the NMI security policy, the NMIServer may either be locally managed or remotely managed by DSMP and an NTMS.

B.8.3 StampServer

The TT StampServer (TSS) is a tamper resistant PCIv2.1 compliant card available with both NT and Solaris drivers. The card is designed to meet FIPS 140-1 L3 physical security requirements. All sensitive cryptographic calculations are performed by a dedicated hardware token that is validated to FIPS 140-1 L2. This token contains a high quality random number generator which is designed to meet FIPS 140-1 L3. The TSS retains all configuration parameters in non-volatile memory elements. The unit contains a high quality clock which will maintain time output specifications through a host server reboot. The StampServer also maintains an audit log of transactions with upper stratum clocks, alarms and operator actions.

The TT StampServer provides IETF-compliant time stamps with Datum Trusted Time Temporal Tokens embedded as an extension to the host Time Stamp Authority server. The TSS’s Trusted Time is guaranteed to be within 100 milliseconds of Coordinated Universal Time (UTC).

B.8.4 Trusted Time API Software Development Kit

The Trusted Time API Software Development Kit (TTAPI SDK) is used by software applications developers to enable their applications to obtain and use Time Stamp Tokens from a TSS-equipped server. Applications using the TTAPI will range from IETF-compliant Time Stamp Authorities to general e-commerce applications. Some applications may use all of the TTAPI capabilities while others will use only a subset.

Functions provided by the TTAPI include:

1. TSS configuration;
2. Cryptographic parameter and key generation;

3. Key backup;
4. Creation and export of Time Stamp Tokens; and
5. TSS test.

The API should is written in C for maximum portability, and it supports operation on Windows 9x and NT platforms and Sun Solaris 2.x.

TTAPI is fully defined in Datum TTAPI SDK Programmers Manual.

B.8.5 Network Time Management System

The Network Time Management System (NTMS) is a standalone command and control platform which provides complete secure management of the remote TTI products. The NTMS is based on a Sun UltraSPARC and it must reside behind a network firewall to protect it against external attack. The NTMS is designed to both manage the TTI products at the device level as well as to provide application layer functionality. GUI driven device level management includes such functions as device initialization, remote configuration, and device test. Application layer functionality includes such actions as Temporal Token validation and audit trail maintenance. A significant feature of the NTMS is that the device management is secure so that the TTP may be remotely monitored and controlled without the risk of spoofing or undetected data alteration. This capability greatly reduces cost of operation for the TTI. Functionally, the elements comprising the NTMS are shown in Figure B-4.

The NTMS Element Manager (EM) functionality is used to configure and manage all the TTPs in a particular TTI. The EM features a GUI interface to provide a simple TTP management and configuration interface for the NTMS operator.

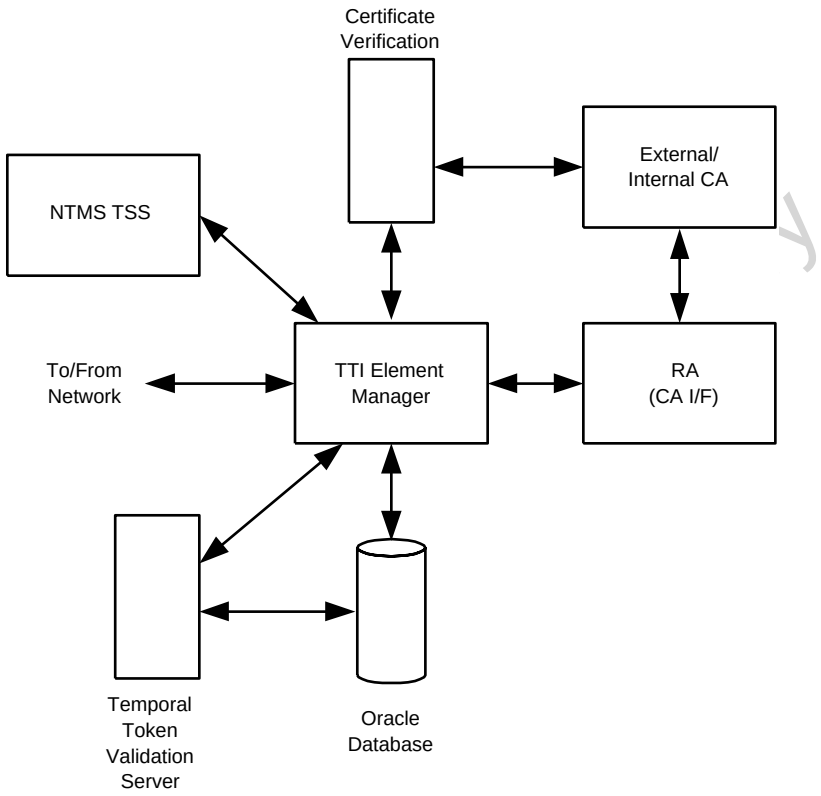

FIGURE B-4: NTMS Functional Elements

When a TTP element is installed, it is contacted by the NTMS. The NTMS assigns the TTP to its respective TTI stratum level and also assigns the appropriate policy information for the TTP. As the TTP's stratum level and policy information will be embedded in its certificate, the NTMS sends the TTP's configuration information and its public key to the RA for transmission to the CA. This portion of the NTMS is designed to be modified so that many third party CA/RA interfaces can be supported. If the RA is physically separate from the platform housing the NTMS, the information sent between the NTMS and RA will be secured by TLS/SSL or IPSEC.

The NTMS also performs Temporal Token verification When a request is received to verify a TT (TTVerReq), the EM will validate the timing chain from the TSS to the NMIServer, or if none, the Stratum 1 TMC. This validation will also include validation of the TTP certificates, CRLs, and applicable policies. Once the validity or invalidity of the TT is determined, the EM will return a digitally signed message indicting this status to the requester.

New Datum software releases are made operational by the NTMS. When Datum makes a new software release applicable to the fielded TTP, the NTMS validates the DFCA digital signature on the received software release. If that signature matches, it then resigns the release using its private key and enables its download to the applicable TTP elements.

The NTMS uses a Chrysalis Luna 2 tamper-resistant cryptographic token to generate and store its private signature key and to perform sensitive calculations. This card contains an RNG which is designed to meet FIPS 140-1 Level 3. The token, itself, has a Level 2 rating. The NTMS is designed to meet FIPS 140-1 Level 3 requirements for unique operator passwords and role separation (i.e., network administration functions are separated from security functions by operator roles). The following roles are provided: System Administrator, Network Manager, Security Officer, and, if required, CA Administrator. The System Administrator is responsible for the configuration and administration of the computing platforms (except the CA) which are used in the NTMS. The Network Manager is responsible for the configuration and day-to-day operation of the TTI. The Security Officer is responsible for items which can alter the trust/security of the TTI. These items include: key generation parameter selection; key/certificate lifecycle; TT and TST policies; inclusion of TTP in a TTI; software download approval; etc. If the internal CA is used, then a CA Administrator role must be created and assigned. The CA Administrator is responsible for CA root generation and rollover, CRL publication, and CA policy.

All operator actions are logged with the operator ID in an audit trail. All NTMS actions (i.e., software download approval; element management functions; and communications with NMIServer operators) are also logged in a audit trail. The NTMS audit trails and those retrieved from the fielded TTP devices (including TCCerts) are written onto CD-ROMs for permanent storage. All data written to CD-ROM is time stamped using the NTMS' TSS.

B.9 Certification Authority

Two types of CAs are available to a TTI: an internal NTMS CA or an external (i.e., separate platform) third party CA. For test sites and small TTI operations, the simple NTMS CA may be sufficient. For large deployments requiring a more fully featured CA, a third party CA is recommended.

The third party CA product or service must support IETF RFC 2459 and either PKCS 7 and 10 or IETF RFCs 2510 and 2511. This CA must support the TTI naming conventions and TTI policy fields in certificate creation. Sensitive data exchanged between the RA and the CA must be authenticated at a minimum.

For the TTI to operate as an unimpeachable source of timing information, the CA/RA must meet stringent security and operational criteria. Access to the CA/RA must be physically secured to prevent unauthorized access. Care must be taken so that the private signature key is not compromised by EMC attack. The CA/RA operating staff must be comprised of trusted individuals with audit records created and maintained for their actions. Individual passwords must be required of each CA/RA operator so that individual actions may be tracked and audited.

Certain CA features are not important for TTI operation. Directory support is not a critical selection factor since the TTP will not be requesting certificates from a directory. Cross certification support is not important as this is not used in the current TTI architecture.

Please note that the exact operational details and features for the TTI CA/RA will depend upon the third party commercial products or services selected.

B.10 TTI Cryptographic and Public Key Infrastructure Elements

B.10.1 Key Agreement Algorithm

Diffie-Hellman is used as the key agreement mechanism for the Datum Secure Network Time Protocol (DS/NTP) and the Datum Secure Management Protocol (DSMP). Common system-wide Diffie-Hellman parameters, generated per ANSI X9.42, are used in the TTI. The modulus length is 1024 bits. Mutual authentication of communicating parties is performed using certificates.

B.10.2 Hash Algorithm

The United States Secure Hash Algorithm – 1 (SHA-1), as defined in FIPS 180-1, is used for all hashing operations required in the TTP.

B.10.3 Hash-based Message Authentication Algorithm (HMAC)

SHA-1 based HMAC, as defined in RFC 2104, is used for high performance authentication of large data quantities and time-sensitive data.

B.11 Digital Signature Methods

The TTI supports two digital signature methods: the FIPS 186-1 Digital Signature Standard (DSS) and the ANSI X9.31 RSA Digital Signature. For DSS, common signature parameters are used in all TTP. These parameters have been generated per FIPS 186. For RSA signatures, the parameters are generated per ANSI X9.31. Both signature schemes use 1024-bit modulus lengths.

B.12 TTI X.509 Certificate Definition

ITU-T X.509v3 certificates and X.509v2 certificate revocation lists are used in the TTI. All certificates contain only signature keys.

- Versionv3
- Serial Numberassigned by CA; unique to certificate
- SignatureSignature algorithm ID and parameters
- IssuerCA name (Distinguished Name)
- ValidityGeneralized Time
- SubjectSubject Name
- Sub Pub KeyDSS public key and parameters belonging to Datum TTP (TSS/TMC). Key pair generated by TTP itself
- Issuer Unique IDOptional CA specific field; not required for TTI
- Sub Unique IDOptional CA specific field; not required for TTI
- Extensions

- Key UsageKeyUsage is set to id-ce-keyUsage-digitalSignature. This extension is flagged critical. This extension is used in NMIServer, NTMS, TMC and TSS device certificates. This extension is also incorporated in and flagged critical in the TSA certificate (see below). For CA root certificates, such as those for the DFCA and the TTI CA, KeyUsage is set to id-ce-keyUsage-keyCertSign. This extension is flagged critical. This extension identifies the subject key as suitable for verifying signatures on certificates.
- Certificate PolicyCurrently unassigned.
- Extended Key UsageKeyPurposeID set to id-kp-timeStamping. This extension is flagged critical. This extension is only used for the TSA certificates. These certificates may be issued by the TTI CA or a third party CA for use by the TSS embedded in a TSA.

Three types of certificates are used in a TTI. A Datum Factory Certificate is created by the Datum Factory CA for each Datum Trusted Time Product (TTP) during manufacture. This certificate enables the NTMS to authenticate remote fielded TTP during installation. A TTI certificate is created by the TTI CA for each fielded TTP. This certificate is network specific and prevents rogue TTP from operating in another TTI. The last certificate is the TSS-TSA certificate which is the certificate for the TSS's time stamp public key. This certificate may be created by either the TTI CA or the CA associated with the TSA.

For operation in the TTI, each TTP must have a distinguished name so that it may be uniquely identified. Datum's naming structure is aligned with the ITU-T X.501/X.520 standards for distinguished names and is as follows:

Country.Organization.Service.Stratum_Level.Manufacturer.Clock_Serial_Number
where:
Country: Country Name (2 letter ISO country codes)
Organization: Organization Name
Service:Service Type (e.g. , NMI; TTI; TTI CA; Others are TBD.)
Stratum_Level:Position in the Timing Hierarchy (per Figure B-2)
Manufacturer:Clock Manufacturer (e.g., Datum, HP, etc.)
Clock_Serial_Number:Serial Number of Clock

Examples of TTI names are:

- For NIST operating as the NMI with a Datum clock:
- US.NIST.NMI.0.Datum.12345XXX

For the commercial TTI Stratum 3 TSS:

- US.Company.TTI.3.Datum.12345XXX
- For a French Timing Service provided by Bull at the NMI level using a Bull clock:
- FR.Bull.NMI.0.Bull.12345XXX
- The above fields, except Country, are limited to 20 ASCII characters each. The Country field is two ASCII characters.
- The fields are organized in an X.509 certificate format as follows:
- Country: Country Name (2 letter IS 3166 country codes)
- Organization: Organization Name
- Organization Unit Name:Service.Stratum_Level
- Common Name:Manufacturer.Clock Serial Number

- The Organizational Unit Name shall be constructed as a sequence of names per RFC 2459TTI Token, Time Calibration Certificate and Selected Message Definitions

B.13 IETF Time Stamp Token and Messages

The Time Stamp Request and Time Stamp Response message structures and the format of the Time Stamp Token (TST) are defined in the IETF PKIX Time Stamp Protocol Internet Draft. The Datum TT is embedded in the extensions field of the TST.

B.14 Datum Temporal Token

The Temporal Token (TT) is a Datum extension to the IETF-defined Time Stamp Token. The TT contains the Trusted Time, TCCert calibration data, and a location pointer for calibration information validation. The TCCert calibration data is the audit trail information which shows when the TSS time was last measured by a TMC. The location pointer indicates to an application where the TT can be validated.
The Temporal Token format is defined as follows:

```
id-temporalTokenOBJECT IDENTIFIER ::= {TBD}

temporalToken ::= SEQUENCE {
versionInteger {v1(1)},
binaryTimeBinaryPreciseUTCTime,
calPointerCalPointer,
validationPointValidationPoint
}

BinaryPreciseUTCTime ::= Integer
--the number of 2-32 seconds since 1 January 1900.
--The reason for this construction is to match the Network Time
--Protocol encoding of 32 bit seconds and 32 bit fractional
--seconds since 1 Jan 1900. The integer encoding is octets in
--twos complement. Until the year 19332, time will be
--represented in 9 octets. The LSB of the 5th octet is units of
--seconds. The lower 4 octets are fractional seconds and the
--upper 5 octets are seconds. Since the encoding is in twos
--complement and the 32nd bit is currently set, the 9th octet is
--currently all zeros. The first 8 octets will match the NTP
--protocol, however NTP rolls over in 2036 but this encoding
--will never rollover. 9 octets will suffice mid way through
--the year 19332. After that more octets can be added.

CalPointer::= SEQUENCE {
        upperClockUpperClockID,
```

```
    --This is the name of the TMC providing the
    --currently valid TSS TCCert calibration permit.
lowerClockLowerClockID,
     --This is name of the TSS in the currently
    --valid TCCert calibration permit.
    tCCertTimeNTP Time,
    --This is the time of the currently valid calibration
    --permit.
     --upperClock + lowerClock + tCCertTime form the
     --pointer used to access Calibration Log data from the
     --NTMS.
}

ValidationPoint ::= SEQUENCE {
   accessMethod     OBJECT IDENTIFIER,
   accessLocation    GeneralName }
--This data is used to specify the access method and location
--of the validation entity or service (e.g., validation server
--or NTMS)

GeneralNames ::= SEQUENCE SIZE (1..MAX) OF GeneralName

GeneralName ::= CHOICE {
   otherName         [0] INSTANCE OF OTHER-NAME,
   rfc822Name        [1] IA5String,
   dNSName           [2] IA5String,
   x400Address       [3] ORAddress,
   directoryName     [4] Name,
   ediPartyName      [5] EDIPartyName,
   uniformResourceIdentifier  [6] IA5String,
   iPAddress         [7] OCTET STRING,
   registeredID      [8] OBJECT IDENTIFIER
}

OTHER-NAME ::= TYPE-IDENTIFIER
```

B.15 Temporal Token Verification Request and Response Formats

A Temporal Token Verification Request (TTVerReq) is issued to the NTMS by an application that wishes to verify the validity of a TT contained in a TST.

B.16 Temporal Token Request Format

Clients use the Temporal Token Verification Request (TTVerReq) to request an NTMS to verify a Temporal Token. The format of the TTVerReq message is:

```
TTVerReq ::= SEQUENCE {
Version             Integer {v1(1)},
ttInfo              TTInfo,
-- TTInfo from time stamp that requires verification including policy
        validateAllPaths  [0] BOOLEAN {FALSE},
        -- if this is TRUE, the NTMS should validate all TCCCert paths
        -- that exist for a TSS in the time in question.
        returnProof[1] BOOLEAN {FALSE}
        -- if this is TRUE, the response will include a TCCert path
        -- that was verified by the validation server.
}
```

- This request contains the Temporal Token (TT) that exists in the time stamp. Options for the request include validateAllPaths and returnProof. The validateAllPaths option tells the NTMS to find and validate all TCCert paths that exists for a particular time stamp. Multiple paths are probable as each TTP can be certified by one or more higher level clocks at overlapping intervals.
- The returnProof option requests the NTMS to return all the TCCerts that were examined during the validation process. Whenever a TT validation is performed, the NTMS also verifies the certificate revocation lists for the time in question to verify that the TTP device certificates were valid.

B.17 Temporal Token Response Format

The TT Validation Server will return a TT Verification Response (TTVerResp), confirming or denying the validity of the TT, to the requesting application. The format of the TTVerResp message is:

```
TTVerResp ::= SEQUENCE {
Version             Integer {v1(0)},
ttInfo              TTInfo,
-- ttInfo from TTVerReq
result                      TTVPResult
-- results of verification
}

TTVPResult ::= CHOICE {
failureCode                 Integer,
```

-- failure code is present if the verification failed
validationData VerificationData
--validationData contains the information to support the
--successful TT validation
}

VerificationData ::= SEQUENCE {
validChainCount Integer,
tcCertChainsTCCertChains OPTIONAL
--this must be present if the client requested proof of
--validation.
--If the request indicates that all cert chains should be
--validated, then this list must include all of the chains
--that were found.
crls CertRevLists
--contains CRL IDs for CRLs pertaining to TTP used to produce
--the TCCert chains.
}

TCCertChains ::= SET {
tcCertChain TCCertChain
}

TCCertChain ::= SET {
tcCert TCCert
}

CertRevLists ::= SET {
crl TTICRL
}

- The TTVerResp is a signed data unit to provide integrity of the NTMS' testimony of the validity of the verification results. The TTVerResp is returned as the encapsulated content of the SignedData PDU:

```
SignedData ::= SEQUENCE{
       versionCMSVersion,
       digestAlgorithmsDigestAlgorithmIdentifiers,
       encapContentInfoEncapsulatedContentInfo,
       certificates[0] IMPLICIT CetificateSet OTIONAL,
       crls[1] IMPLICIT CertificateRevocationLists OPTIONAL
       signerInfosSignerInfos }

SignerInfos ::= SET of SignerInfo

EncapsulatedContentInfo ::= SEQUENCE {
```

```
       eContentTypeContentType,
       eContent[0] EXPLICIT OCTET STRING OPTIONAL }

ContentType ::= OBJECT IDENTIFIER

The fields of type EncapsulatedContentInfo have the following meanings:

eContentType is an object identifier that uniquely specifies the content type. For a    Tem-
poral Token Verification Response, it is defined as:

id-ct-TTVerResp OBJECT IDENTIFIER ::= {id-ct TBD}

with:

id-ct     OBJECT IDENTIFER ::= {id-smime 1}
id-smimeOBJECT IDENTIFER ::= {iso (1) member-body (2) us (840) rsadsi (113549) pkcs (1)
                       pkcs-9 (9) 16}
```

- eContent is the content itself, carried as an octet string. The eContent is the DER-encoded value of TTVerResp.

B.18 TCCert Definition

When a higher level clock measures the offset of a lower clock and finds it within specification, a Time Calibration Certificate or TCCert is created to record this fact. The TCCert consists of the following fields:

- Version
- Upper Clock ID:
- Country.Organization.Service.Service_Level.Manufacturer.Clock_Serial_Number
- Lower Clock ID:
- Country.Organization.Service.Service_Level.Manufacturer.Clock_Serial_Number
- TCCert Time:NTP Time (represented as an ASN.1 Integer)
- Dispersion:microseconds (represented as an ASN.1 Integer)
- Delaymicroseconds (represented as an ASN.1 Integer)
- TCCert Expire Time:NTP Time (represented as an ASN.1 Integer)
- Upper Clock Sig Alg:Upper Clock's signature algorithm OID
- Upper Clock Sig Params:DSA signature parameters of Upper Clock
- Upper Clock Signature:DSA signature across above fields
- Lower Clock Sig Alg:Lower Clock's signature algorithm OID
- Lower Clock Sig Params:DSA signature parameters of Lower Clock
- Lower Clock Signature:DSA signature across above fields

Additional explanations of selected fields:

- Version identifies the version of the TCCert.
- Lower and Upper Clock Signature Parameters are those contained in the X.509 certificate definition. For the first release of the TTP, these fields will be NULL since system-wide parameters will be used.
- Dispersion provides the measure of the offset of the lower clock from UTC.
- Delay is the propagation delay between the host and peer as specified by Mills in RFC1305.
- TCCert Expiration Time is a policy item determined by the NTMS. The default expiration time for a TCCert is 72 hours. The Clock Offset field is determined by the Upper Clock.

The ASN.1 representation for the TCCert is shown below:

```
SignedData ::= SEQUENCE{
version     CMSVersion,
digestAlgorithmsDigestAlgorithmIdentifiers,
encapContentInfoEncapsulatedContentInfo,
certificates[0] IMPLICIT CetificateSet OTIONAL,
crls              [1] IMPLICIT CertificateRevocationLists OPTIONAL
signerInfos  SignerInfos }

SignerInfos ::= SET of SignerInfo

EncapsulatedContentInfo ::= SEQUENCE {
eContentTypeContentType,
eContent[0] EXPLICIT OCTET STRING OPTIONAL }

ContentType ::= OBJECT IDENTIFIER

      -- The fields of type EncapsulatedContentInfo have the following
-- meanings:
      --
      -- eContentType is an object identifier that uniquely specifies the
-- contenttype. For a TCCert, it is defined as:

      id-ct-TCCert OBJECT IDENTIFIER ::= {id-ct TBD}

      -- with:

      id-ctOBJECT IDENTIFER ::= {id-smime 1}
id-smimeOBJECT IDENTIFER ::= {iso (1) member-body (2) us (840) rsadsi (113549)
            pkcs (1) pkcs-9 (9) 16}

      -- eContent is the content itself, carried as an octet string.
      -- The eContent content type shall have ASN.1 type TTInfo.

      -- The TCCert is the content of the SignedData type. Because we are
```

```
-- "double signing" the data, the Upper Clock will create the TCCert,
-- sign it, and output a signed data PDU. The lower clock will verify
-- the data in the TCCert, verify the signature of the SignedData and
-- embed the SignedData(TCCert) from the upper clock into a new
-- SignedData type. So it should look like this:

-- SignedData(SignedData(TCCert, Upper Clock), Lower Clock)
--
-- where SignedData() == SignedData(Content, Signer)
--
--

TCCert {OID TBD} DEFINITIONS ::=
BEGIN

TCCert ::= SEQUENCE {
certInfoTimingInformation,
expirationINTEGER
};

TimingInformation ::= SEQUENCE {
ntpTimeNTPTime,
      offsetINTEGER,
delay       INTEGER
};

NTPTime ::= INTEGER;

END
```

Notes

Appendix C: Trusted Time Acronyms

Table C-1: Trusted Time™ Acronyms

Acronym	Definition
APC	Application Certificate
API	Application Program Interface
CA	Certification Authority or Certificate Authority
CRL	Certificate Revocation List
CRM	Certificate Request Message
DFC	Datum Factory Certificate
DFCA	Datum Factory Certificate Authority
DSA	Digital Signature Algorithm specified in DSS
DSS	Digital Signature Standard (FIPS 186)
DS/NTP	Datum Secure Network Time Protocol
ENMTMS	Element Manager
FIPS	Federal Information Processing Standard
GPS	Global Positioning System
IETF	Internet Engineering Task Force
NIST	National Institute of Standards and Technology
NMI	National Measurement Institute(s)
NMIServer	National Measurement Institute Server
NOC	Network Operations Center
NTMS	Network Time Management System
NTP	Network Time Protocol (RFC 1305)
OCSP	Online Certificate Status Protocol
OCSPROCSP	Responder
OEM	Original Equipment Manufacturer
OID	Object Identifier
PKI	Public Key Infrastructure
PLB	Private Label Branch
PSTN	Public Switched Telephone Network
RA	Registration Authority
SNMP	Simple Network Management Protocol
SSL	Secure Sockets Layer
TCCert	Time Calibration Certificate
TLS	Transport Layer Security
TMC	Trusted Master Clock
TPC	Third Party Certificate
TPCA	Third Party Certification/Certificate Authority
TSA	Time Stamp Authority
TSR	Time Stamp Request
TSS	TT StampServer
TST	Time Stamp Token
TT	Datum Temporal Token
TTAPI	Trusted Time API
TTDS	Trusted Time Distribution Service
TTI	Trusted Time Infrastructure
TTP	Trusted Time Product
UDP/IP	User Datagram Protocol/Internet Protocol
UTC	Coordinated Universal Time

Index