Time Stamp Option Pack

TSOP SDK v8.1.0 User guide

18 October 2024

Table of Contents

1. Introduction

The Entrust nShield Time Stamp Option Pack (TSOP) Software Development Kit enables you to integrate an Entrust nShield Time Stamp Server™ (TSS) PKIX-compliant time-stamp into an application, file, log, or transaction. The TSOP SDK documentation includes:

- Instructions for installing the SDK.
- Guidelines for API development.
- Sample code.

This section provides instructions for installing the SDK and example applications on various operating systems. It also provides information about the locations where the software and example applications are installed.

1.1. Overview of the TSOP SDK

The TSOP SDK provides Java and C libraries implementing a set of functions that enable you to develop time-stamping applications or integrate time-stamping into existing applications. The SDK also includes sample code and a set of test applications.

The functions provided by the API include:

- Encoding and decoding time-stamp requests.
- Decoding time-stamp responses.
- Decoding time-stamp tokens.
- Generating digests.
- Verifying digital signatures of time-stamp tokens.

The API is available in Java language classes and in C language libraries.

The C language API is available as both a 32-bit and 64-bit DLL and as a 64-bit Linux static library.

The API supports the cryptographic mechanisms that create hashes, validate tokens, certify signatures, and establish TSS sessions.

The sample code and applications provided in the SDK include examples of:

- Generating digests and building time-stamp requests.
- Using the Time-Stamp TCP/IP protocol to submit requests and get responses from a TSS.

- Decoding and verifying time-stamp responses.

2. Getting started

2.1. Install the TSOP SDK

The following sections explain how to install the SDK on Windows and Linux.

2.1.1. Windows installation

To install on a Windows workstation, follow these steps:

1. Ensure you have uninstalled any previously installed version of the SDK.
2. Insert the SDK disk (named `TSOP-SDK-x.xx.xx`, where *x.xx.xx* represents the disk's version number) into your CD/DVD drive.
3. Run the executable file `win\setup.msi`, and follow the onscreen instructions.

2.1.2. Linux installation

The SDK is distributed as two compressed `.tar` files. To install on a Linux workstation, follow these steps:

1. Ensure you have uninstalled any previously installed version of the SDK.
2. Unpack the files from the root directory using the following commands:
   - For C:

     ```
     tar -xvf TSOP-SDK-x.xx.xx/linux/c/devel.tar
     ```

   - For Java:

     ```
     tar -xvf TSOP-SDK-x.xx.xx/linux/java/devel.tar
     ```

In these commands, *x.xx.xx* represents the disk's version number.

2.2. Locations of the installed software and examples

On Windows, the Java and C versions of the SDK software are installed, respectively, in:

- `%NFAST_HOME%\java\dsesdk`
- `%NFAST_HOME%\c\dsesdk`

On Linux, the Java and C versions of the SDK software are installed, respectively, in:

- `/opt/nfast/java/dsesdk`
- `/opt/nfast/c/dsesdk`

2.3. Java examples

The following example Java applications are supplied on each operating system:

Example applications	Use
`TtiTest.java` `HttpTest.java`	These applications get a time-stamp from a server and print information from the time-stamp to the screen. `TtiTest.java` uses the TCP/318 socket-based protocol to get the time-stamp from the server. `HttpTest.java` uses the HTTP protocol to get the time-stamp from the server.
`TtiStress.java` `HttpStress.java`	These applications provide information on: • Total running time in milliseconds. • Total count of successful time-stamps received. • The count of time-stamps received during this second. • Average time-stamps per second since program start-up. • Total count of network retries followed by the count of double, triple, and quadruple retries. • Total number of ignored exceptions.

The contents of the `java/dsesdk/` directory tree is the same for all operating systems (the path descriptions here use Unix-style directory separators, but you can substitute Windows-style directory separators as appropriate).

File / directory	Description
`classes/tti.jar`	This file is the `.jar` file that contains all the public classes from the Java version of the SDK.
`docs/`	This directory contains the Javadoc documentation for the SDK Java classes.
`examples/sample/`	This directory contains example applications: • `TtiTest.java` • `HttpTest.java` • `TtiStress.java` • `HttpStress.java`.

File / directory	Description
examples/util/	This directory does not contain example applications but rather a collection of example classes, in the form of sample code that you can use and modify.
docs-util/	This directory contains Javadoc documentation generated from the source example classes in the examples/util/ directory.
classes/util/util.jar	This file is the compiled .jar file from the sample class files in the examples/util/ directory.

ℹ See the example help message for details of parameters that can be varied.

2.4. C examples

The following sections discuss the C application examples available for Windows and Linux.

ℹ See the example help message for details of parameters that can be varied.

2.4.1. C examples for Windows

The contents of `%NFAST_HOME%\c\dsesdk\` provide the source and compiled binaries for two sample applications:

Example applications	Use
ttitest	This application uses the TCP/318 socket-based protocol to get the time-stamp from the server and then prints the information from the time-stamp to the screen.
iptsdemo	This application (IP Time-Stamp Demo), is a Windows GUI time-stamp application. There is no version of IP Time-Stamp Demo for Linux.

In `%NFAST_HOME%\c\dsesdk\` you will find:

File / directory	Description
vs2017-32\lib\ vs2017-64\lib\	These directories contain the DLL, LIB, and PDB files for the SDK, for 32-bit and 64-bit respectively.
vs2017-32\include\ vs2017-64\include\	These directories contain the header files for the SDK, for 32-bit and 64-bit respectively.

File / directory	Description
`prebuilt\`	This directory includes prebuilt versions of the example applications `ttitest.exe` and `iptsdemo.exe`.
`bin\`	This is the output directory for the example projects that Visual Studio builds. It also contains prebuilt versions.
`ttitest\`	This directory contains the source for the `ttitest` application.
`iptsdemo\`	This directory contains the source for IP Time-Stamp Demo, `iptsdemo`.

2.4.2. C examples for Linux

Linux installations provide the source, compiled binaries, and Makefiles for the example C application `ttitest`.

Example application	Use
`ttitest`	This application uses the TCP/318 socket-based protocol to get the time-stamp from the server and then prints the information from the time-stamp to the screen.

On Linux, the `/opt/nfast/c/dsesdk` directory contains:

File / directory	Description
`gcc/include/`	This directory contains header files for the SDK.
`gcc/lib/libtti.a`	This is the library file for the SDK.
`gcc/examples/`	This directory contains the compiled binary executables, source code, and Makefiles for the `ttitest` application.

3. C language API functions and specifications

This section describes the API functions and specifications.

> The C language API is currently provided in Windows and Linux versions.

3.1. Functional overview

The SDK can be used to obtain a time-stamp from a TSS with four basic steps:

- Create an encoded request
- Submit the request to the TSS
- Decode the result
- Verify the integrity of the time-stamp

To obtain a time-stamp from a TSS:

1. Using the `TTI_SHA*` functions, generate a digest of the data to be time-stamped.
2. Generate a nonce for the request. A nonce is a large random number that protects the request against replay attacks.
3. Populate a time-stamp request structure, `TTI_TSQ_Ex`, with the digest, nonce, and other relevant information.
4. Call `TTI_EncodeTSQ_Ex` to create an ASN.1 encoded version of the request.
5. Submit the encoded request to a TSS. The request will usually be submitted via a TCP/IP connection.
6. Unpack the response returned by the TSS by calling `TTI_UnpackTSR`. This will verify and remove the transport specific headers from the response.
7. Call `TTI_GetTSR_Status` to verify the response successfully produced a time-stamp token.
8. Call `TTI_GetTSR_EncodedToken` to get the time-stamp token from the response. This produces an encoded time-stamp token.

   The encoded time-stamp token is a PKCS #7 SignedData object and the signature can be verified with any cryptographic toolkit that supports PKCS #7.

   The `TTI_VerifyTST_SignatureEx` function can also be used to verify the time-stamp signature.

9. Verify the time-stamp token signature. Use `TTI_VerifyTST_SignatureEx` or a

cryptographic toolkit that supports PKCS #7.

10. Call TTI_GetTST_TSTInfoEx to populate a TTI_TSTInfoEx structure with the contents of the time-stamp token.
11. Verify that:
    - The value in the time-stamp token (TST) matches the values in the timestamp request (TSQ).
    - The message imprint matches.
    - The nonce matches.
    - If the request included a specific policy identifier, this should match.
    - The time contained in the time-stamp is reasonably close to the current system time (only perform this check if you can trust the system time to be relatively accurate).

3.2. Functions and specifications

The following sections document the API functions and their specifications.

3.3. API functions

The following sections explain the API specifications.

3.3.1. TTI_EncodeTSQ_Ex

Uses the information in a TTI_TSQ_Ex structure to create an encoded time-stamp request.

```
int TTI_EncodeTSQ_Ex(
    const TTI_TSQ_Ex *     pTSQ,
    byte *                 encodedReq,
    size_t *               encodedReqLen,
    TTI_TransportFormat    transportFormat  );
```

3.3.1.1. Parameters

pTSQ	[in] Points to a populated TTI_TSQ_Ex structure. The information in this structure is used to create the encoded time-stamp request. The TTI_TSQ_Ex structure supports nonce and serial numbers up to 40 bytes.
encodedReq	[out] Points to a buffer to receive the encoded time-stamp request.

`encodedReqLen`	[in/out] Points to a value specifying the size, in bytes, of the `encodedReq` buffer. When the function returns, this value contains the size, in bytes, of the encoded time-stamp request.
`transportFormat`	[in] A flag indicating which type of additional transport encoding should be included in the request.

Currently defined format types are:

`TTI_RAW`	Returns the encoded time-stamp request with no headers and no special encoding.
`TTI_TCP`	Returns the encoded time-stamp request prepended with a five-byte TCP header.
`TTI_HTTP`	Returns the time-stamp request encoded as an HTTP encoded request.
`TTI_SMTP`	Reserved for future use.

3.3.1.2. Return values

If this function succeeds, the return value is zero (`TTI_SUCCESS`).

If this function fails, the return value is a nonzero error code.

`TTI_INVALID_PARAMETER`	`pTSQ` and `encodedReqLen` must not be NULL. `transportFormat` must be a valid `TTI_TransportFormat` value.
`TTI_BUFFER_TOO_SMALL`	The size indicated by `encodedReqLen` is too small. When the function returns, the required size is returned in the value pointed to by `encodedReqLen`.
`TTI_TSR_ASN_ERROR`	An unexpected ASN error occurred. To get the ASN error code, call `TTI_GetLastAsnError`.
`TTI_NOT_SUPPORTED`	`TTI_SMTP` is not supported at this time.

3.3.1.3. Remarks

This function creates an encoded time-stamp request that includes the information supplied in the `TTI_TSQ_Ex` structure. The request is formatted according to version one of the PKIX Time-Stamp protocol. This function will also encode the request for a particular transport mechanism. At this time, two transport format options are supported: `TTI_RAW` and `TTI_TCP`. `TTI_RAW` returns the encoded time-stamp request with no headers and no special encoding. `TTI_TCP` returns the request with a five-byte TCP header prepended. This header includes the size of the request and a flag byte set to zero (`tsaMsg`).

When this function is used with the transportFormat set to TTI_TCP, the resulting encoded time-stamp request may be submitted directly to a TSS via a TCP socket connected to port 318 of the server.

3.3.2. TTI_DecodeTSQ_Ex

Decodes an encoded time-stamp request and writes the information into a TTI_TSQ_Ex structure.

```
int TTI_DecodeTSQ_Ex(
    TTI_TSQ_Ex *    pTSQ,
    const byte *    encodedReq,
    size_t          encodedReqLen  );
```

3.3.2.1. Parameters

pTSQ	[out] Points to a TTI_TSQ_Ex structure that receives information from the encoded time-stamp request. The TTI_TSQ_Ex structure supports nonce and serial numbers up to 40 bytes.
encodedReq	[in] Points to a buffer that contains an encoded time-stamp request.
encodedReqLen	[in] Specifies the size, in bytes, of the encoded time-stamp request.

3.3.2.2. Return values

If this function succeeds, the return value is zero (TTI_SUCCESS).

If this function fails, the return value is a nonzero error code.

TTI_INVALID_PARAMETER	pTSQ and encodedReqLen must not be NULL.
TTI_ASN_ERROR	Received unexpected results from an ASN function.
TTI_TSR_ASN_ERROR	An unexpected ASN error occurred. To get the ASN error code, call TTI_GetLastAsnError.

3.3.2.3. Remarks

This function can be used to decode a time-stamp request that was encoded with TTI_EncodeTSQ_Ex. It is potentially useful if the original TTI_TSQ_Ex structure that was used to create the encoded request is not available.

3.3.3. TTI_UnpackTSR

Converts a transport-specific response into a raw time-stamp response by removing transport-specific headers.

```
    int TTI_UnpackTSR(
        byte *              encodedResp,
        size_t *            encodedRespLen,
        int *               responseCode,
        TTI_TransportFormat transportFormat );
```

3.3.3.1. Parameters

encodedResp	[in/out] Points to a buffer that contains a transport-specific response. When the function returns, this buffer will contain a raw time-stamp response.
encodedRespLen	[in/out] Points to a value specifying the size, in bytes, of the transport-specific response. When the function returns, this value contains the size, in bytes, of the raw time-stamp response.
responseCode	[out] Points to an integer to receive the response code from a TTI_TCP response.
transportFormat	[in] The transport mechanism that produced this response.

Currently defined format types are:

TTI_RAW	Encoded time-stamp response with no headers and no special encoding.
TTI_TCP	Encoded time-stamp response prepended with a five-byte TCP header.
TTI_HTTP	Encoded time-stamp response with HTTP response headers.
TTI_SMTP	Reserved for future use.

3.3.3.2. Return values

If this function succeeds, the return value is zero (TTI_SUCCESS).

If this function fails, the return value is a nonzero error code.

TTI_INVALID_PARAMETER	encodedResp, encodedRespLen, and responseCode must not be NULL. transportFormat must be a valid TTI_TransportFormat value.
TTI_RESPONSE_SIZE_MISMATCH	The size from the TCP header did not match the actual size of the response. The expect size from the TCP header is returned in the value pointed to by encodedRespLen.

TTI_TSA_UNEXPECTED_RESPONSE	The response flag in the TCP header contained an unexpected value. The only expect value is 5 (finalMsgRep). The actual value is returned in the value pointed to by responseCode.
TTI_NOT_SUPPORTED	TTI_SMTP is not supported at this time.

3.3.3.3. Remarks

This function removes the header and special encoding that is added to a time-stamp response by the transport mechanism. The transport format options supported are:

- TTI_RAW (only supported for completeness)
- TTI_TCP.

If the format is TTI_RAW, this function simply returns TTI_SUCCESS without doing any processing because none is needed. If the format is TTI_TCP, this function will verify and remove the TCP header from the time-stamp response.

3.3.4. TTI_GetTSR_Status

Decodes a time-stamp response and writes the information into a TTI_PKIStatusInfo structure.

```
int TTI_GetTSR_Status(
    TTI_PKIStatusInfo *   pPKIStatusInfo,
    const byte *          encodedResp,
    size_t                encodedRespLen  );
```

3.3.4.1. Parameters

pPKIStatusInfo	[out] Points to a TTI_PKIStatusInfo that receives information from the encoded time-stamp response.
encodedResp	[in] Points to a buffer that contains an encoded time-stamp response.
encodedRespLen	[in] Specifies the size, in bytes, of the encoded time-stamp response.

3.3.4.2. Return values

If this function succeeds, the return value is zero (TTI_SUCCESS).

If this function fails, the return value is a nonzero error code.

TTI_INVALID_PARAMETER	pPKIStatusInfo and encodedResp must not be NULL.
TTI_ASN_ERROR	Received unexpected results from an ASN function.
TTI_TSR_ASN_ERROR	An unexpected ASN error occurred. To get the ASN error code, call TTI_GetLastAsnError.

3.3.4.3. Remarks

This function retrieves the status information from a time-stamp response. This status information indicates whether a time-stamp token was issued in the response. If a time-stamp token was not issued, the status information indicates the reason for the failure.

3.3.5. TTI_GetTSR_EncodedToken

Retrieves the encoded time-stamp token from an encoded time-stamp response.

```
    int TTI_GetTSR_EncodedToken(
        byte *          tokenBuf,
        size_t *        tokenBufLen,
        const byte *    encodedResp,
        size_t          encodedRespLen   );
```

3.3.5.1. Parameters

tokenBuf	[out] Points to a buffer that receives the encoded time-stamp token.
tokenBufLen	[in/out] Points to a value specifying the size, in bytes, of the tokenBufbuffer. When the function returns, this value contains the size, in bytes, of the encoded time-stamp token.
encodedResp	[in] Points to a buffer that contains an encoded time-stamp response.
encodedRespLen	[in] Specifies the size, in bytes, of the encoded time-stamp response.

3.3.5.2. Return values

If this function succeeds, the return value is zero (TTI_SUCCESS).

If this function fails, the return value is a nonzero error code.

TTI_INVALID_PARAMETER	encodedResp and tokenBufLen must not be NULL.
TTI_BUFFER_TOO_SMALL	The size indicated by tokenBufLen is too small. When the function returns, the required size is returned in the value pointed to by tokenBufLen.

TTI_ASN_ERROR	Received unexpected results from an ASN function.
TTI_TSR_ASN_ERROR	An unexpected ASN error occurred. To get the ASN error code, call TTI_GetLastAsnError.
TTI_INVALID_RESPONSE_STATUS	The time-stamp response status was not PKIS_granted or PKIS_grantedWithMods. This indicates that the timestamp response was a failure response, therefore there is no time-stamp token in the response.
TTI_NO_TOKEN_PRESENT	No token was present in the response. This is an unexpected situation and indicates corrupt data.

3.3.5.3. Remarks

A time-stamp response is what a TSS returns for a request. The response contains status information indicating what kind of response this is. If the status information indicates PKIS_granted or PKIS_grantedWithMods, the time-stamp response will also contain a time-stamp token. This function copies the time-stamp token into the supplied buffer. The time-stamp token is a Cryptographic Message Syntax SignedData object (see RFC 3369 [CMS2], RFC 3852 [CMS] and RFC 2315 [PKCS #7]). The time-stamp token represents the data that should be stored for future reference. This is the time-stamp.

3.3.6. TTI_GetTST_TSTInfoEx

Decodes an encoded time-stamp token and writes the encapsulated TSTInfo data into a TTI_TSTInfoEx structure.

```
    int TTI_GetTST_TSTInfoEx(
        TTI_TSTInfoEx *    pTSTInfo,
        const byte *       encodedToken,
        size_t             encodedTokenLen );
```

3.3.6.1. Parameters

pTSTInfo	[out] Points to a TTI_TSTInfoEx structure that receives information from the encoded time-stamp token. The TTI_TSTInfoEx structure supports nonce and serial numbers up to 40 bytes.
encodedToken	[in] Points to a buffer that contains an encoded time-stamp token.
encodedTokenLen	[in] Specifies the size, in bytes, of the encoded time-stamp token.

3.3.6.2. Return values

If this function succeeds, the return value is zero (TTI_SUCCESS).

If this function fails, the return value is a nonzero error code.

TTI_INVALID_PARAMETER	encodedResp and tokenBufLen must not be NULL.
TTI_ASN_ERROR	Received unexpected results from an ASN function.
TTI_TSR_ASN_ERROR	An unexpected ASN error occurred. To get the ASN error code, call TTI_GetLastAsnError.

3.3.6.3. Remarks

This is the core data of a time-stamp token. The TSTInfo is part of the signed data of the time-stamp token and therefore is protected against modification. This function reads and decodes this portion of the time-stamp token and writes the information into a TTI_TSTInfoEx structure.

3.3.7. TTI_GetTST_TSACertEx

Retrieves the encoded TSA certificate from an encoded time-stamp token.

```
int TTI_GetTST_TSACertEx(
    byte *                 certBuf,
    size_t *               certBufLen,
    const byte *           encodedToken,
    size_t                 encodedTokenLen,
    TTI_VerificationModes  verifyModes,
    TTI_SigningCert_Kind * signingCertKind );
```

3.3.7.1. Parameters

certBuf	[out] Points to a buffer that receives the encoded TSA certificate.
certBufLen	[in/out] Points to a value specifying the size, in bytes, of the certBuf buffer. When the function returns, this value contains the size, in bytes, of the encoded TSA certificate.
encodedToken	[in] Points to a buffer that contains an encoded time-stamp token.
encodedTokenLen	[in] Specifies the size, in bytes, of the encoded time-stamp token.
verifyModes	[in] A bitmask identifying how to verify the signing certificate (see below).
signingCertKind	[out] On success, points to a value identifying how the signing certificate was verified (see below).

The supported values for verifyModes are:

TTI_VerifyMode_ESSCertID	Checks the first SHA-1 hash in the ESSCertID list.
TTI_VerifyMode_ESSCertIDv2	Checks the first SHA-2 hash in the ESSCertIDv2 list.
TTI_VerifyMode_ESSCertID | TTI_VerifyMode_ESSCertIDv2	Checks the first SHA-2 hash in the ESSCertIDv2 list. If no SHA-2 hashes are present, checks the first SHA-1 hash in the ESSCertID list.

The supported values for signingCertKind are:

TTI_Verified_SigningCert_ESSCertID	The signing certificate was verified using the SHA-1 hash in the ESSCertID list.
TTI_Verified_SigningCert_ESSCertIDv2	The signing certificate was verified using the SHA-2 hash in the ESSCertIDv2 list.

3.3.7.2. Return values

If this function succeeds, the return value is zero (TTI_SUCCESS).

If this function fails, the return value is a nonzero error code.

TTI_INVALID_PARAMETER	encodedToken and certBufLen must not be NULL, and verifyModes must be a supported combination of TTI_VerificationModes values.
TTI_BUFFER_TOO_SMALL	The size indicated by certBufLen is too small. When the function returns, the required size is returned in the value pointed to by certBufLen.
TTI_INVALID_TST	The content of the encodedToken buffer is not recognized as a valid encoded time-stamp token.
TTI_NO_TSACERT_PRESENT	The encoded time-stamp token did not contain the signing certificate. A time-stamp token will only contain the signing certificate if the time-stamp request specified that the certificate be included.
TTI_ESSCERTID_FAILED	The hash of the Time Attribute certificate did not match a value stored in the signed attributes of the time-stamp token.

3.3.7.3. Remarks

If the encoded time-stamp token contains the signing certificate, this function copies the encoded signing certificate into the supplied buffer. This function does not validate the signature on the time-stamp token. This function verifies the signing certificate based on the verifyModes bitmask described above. The ESSCertID provides cryptographic binding of the time-stamp token to a particular identity certificate, whereas the signature only binds the time-stamp token to the public key. Checking ESSCertIDv2 rather than ESSCertID will succeed only if the TSS is configured to include ESSCertIDv2 in the certificate as per RFC 5816.

3.3.8. TTI_GetTST_TimeAttributeCertEx

Retrieves the encoded Time Attribute Certificate from an encoded time-stamp token.

```
int TTI_GetTST_TimeAttributeCertEx(
    byte *                 certBuf,
    size_t *               certBufLen,
    const byte *           encodedToken,
    size_t                 encodedTokenLen,
    TTI_VerificationModes  verifyModes,
    TTI_SigningCert_Kind * signingCertKind );
```

3.3.8.1. Parameters

certBuf	[out] Points to a buffer that receives the encoded Time Attribute Certificate.
certBufLen	[in/out] Points to a value specifying the size, in bytes, of the certBuf buffer. When the function returns, this value contains the size, in bytes, of the encoded Time Attribute Certificate.
encodedToken	[in] Points to a buffer that contains an encoded time-stamp token.
encodedTokenLen	[in] Specifies the size, in bytes, of the encoded time-stamp token.
verifyModes	[in] A bitmask identifying how to verify the Time Attribute Certificate (see below).
signingCertKind	[out] On success, points to a value identifying how the Time Attribute Certificate was verified (see below).

The supported values for verifyModes are:

TTI_VerifyMode_ESSCertID	Checks the SHA-1 hashes in the ESSCertID list.
TTI_VerifyMode_ESSCertIDv2	Checks the SHA-2 hashes in the ESSCertIDv2 list.
TTI_VerifyMode_ESSCertID | TTI_VerifyMode_ESSCertIDv2	Checks the SHA-2 hashes in the ESSCertIDv2 list. If no SHA-2 hashes are present, checks the SHA-1 hashes in the ESSCertID list.

The supported values for signingCertKind are:

TTI_Verified_SigningCert_None	The Time Attribute Certificate was not verified against any hash in either of the ESSCertID or ESSCertIDv2 lists.
TTI_Verified_SigningCert_ESSCertID	The Time Attribute Certificate was verified using the SHA-1 hash in the ESSCertID list.
TTI_Verified_SigningCert_ESSCertIDv2	The Time Attribute Certificate was verified using the SHA-2 hash in the ESSCertIDv2 list.

3.3.8.2. Return values

If this function succeeds, the return value is zero (TTI_SUCCESS).

If this function fails, the return value is a nonzero error code.

TTI_INVALID_PARAMETER	encodedToken and certBufLen must not be NULL, and verifyModes must be a supported combination of TTI_VerificationModes values.
TTI_BUFFER_TOO_SMALL	The size indicated by certBufLen is too small. When the function returns, the required size is returned in the value pointed to by certBufLen.
TTI_INVALID_TST	The content of the encodedToken buffer is not recognized as a valid encoded time-stamp token.
TTI_NO_TAC_PRESENT	The encoded time-stamp token did not contain the Time Attribute certificate. A time-stamp token will only contain the Time Attribute certificate if the time-stamp request specified that the certificate be included.
TTI_ESSCERTID_FAILED	The hash of the Time Attribute certificate did not match a value stored in the signed attributes of the time-stamp token.

3.3.8.3. Remarks

If the encoded time-stamp token contains the Time Attribute Certificate, this function copies the encoded TAC into the supplied buffer. This function verifies that the time-stamp token was issued under the Time Attribute Certificate before the function returns. This function verifies the Time Attribute Certificate based on the verifyModes bitmask described above. If the Time Attribute Certificate in the encoded time-stamp token is embedded within the set of SignerAttributes and not the certificate list, the TAC will not be verified using ESSCertID or ESSCertIDv2 hashes. This will be indicated by the value TTI_Verified_SigningCert_None being set in signingCertKind. This check is necessary because the certificate list in a time-stamp token is not protected by the signature. The cryptographic binding of a time-stamp to a Time Attribute Certificate is accomplished by including the appropriate hash of the Time Attribute Certificate in ESSCertID or ESSCertIDv2. Both ESSCertID and ESSCertIDv2 are protected by the signature of the time-stamp token. Checking ESSCertIDv2 rather than ESSCertID will succeed only if the TSS is configured to include ESSCertIDv2 in the certificate as per RFC 5816.

3.3.9. TTI_GetTAC_TimingMetricsEx

Decodes an encoded Time Attribute certificate and writes the encapsulated TimingMetrics attribute data into a TTI_TimingMetricsEx structure.

```
int TTI_GetTAC_TimingMetricsEx(
        TTI_TimingMetricsEx *    pTimingMetrics,
        const byte *             certBuf,
        size_t                   certBufLen );
```

3.3.9.1. Parameters

`pTimingMetrics`	[out] Points to a `TTI_TimingMetricsEx` structure that receives information from the encoded Time Attribute certificate.
`certBuf`	[in] Points to a buffer that contains an encoded Time Attribute certificate.
`certBufLen`	[in] Specifies the size, in bytes, of the encoded Time Attribute certificate.

3.3.9.2. Return values

If this function succeeds, the return value is zero (`TTI_SUCCESS`).

If this function fails, the return value is a nonzero error code.

`TTI_INVALID_PARAMETER`	`pTimingMetrics` and `certBuf` must not be NULL.
`TTI_ASN_ERROR`	Received unexpected results from an ASN function.
`TTI_TSR_ASN_ERROR`	An unexpected ASN error occurred. To get the ASN error code, call `TTI_GetLastAsnError`.

3.3.10. TTI_GetTAC_TimingPolicy

Decodes an encoded Time Attribute certificate and writes the encapsulated `TimingPolicy` attribute data into a `TTI_TimingPolicy` structure.

```
    int TTI_GetTAC_TimingPolicy(
        TTI_TimingPolicy *    pTimingPolicy,
        const byte *          certBuf,
        size_t                certBufLen );
```

3.3.10.1. Parameters

`pTimingPolicy`	[out] Points to a `TTI_TimingPolicy` structure that receives information from the encoded Time Attribute certificate.
`certBuf`	[in] Points to a buffer that contains an encoded Time Attribute certificate.
`certBufLen`	[in] Specifies the size, in bytes, of the encoded Time Attribute certificate.

3.3.10.2. Return values

If this function succeeds, the return value is zero (TTI_SUCCESS).

If this function fails, the return value is a nonzero error code.

TTI_NO_TAC_TIMINGPOLICY_PRESENT	The Time Attribute Certificate does not contain Timing Policy information.
TTI_INVALID_PARAMETER	pTimingPolicy and certBuf must not be NULL.
TTI_ASN_ERROR	Received unexpected results from an ASN function.
TTI_TSR_ASN_ERROR	An unexpected ASN error occurred. To get the ASN error code, call TTI_GetLastAsnError.

3.3.11. TTI_CheckTAC_MatchesTSTEx

Verifies that the time-stamp token was issued under the Time Attribute certificate.

```
int TTI_CheckTAC_MatchesTSTEx(
    const byte *           certBuf,
    size_t                 certBufLen,
    const byte *           encodedToken,
    size_t                 encodedTokenLen,
    TTI_VerificationModes  verifyModes,
    TTI_SigningCert_Kind * signingCertKind );
```

3.3.11.1. Parameters

certBuf	[in] Points to a buffer that contains an encoded Time Attribute certificate.
certBufLen	[in] Specifies the size, in bytes, of the encoded Time Attribute certificate.
encodedToken	[in] Points to a buffer that contains an encoded time-stamp token.
encodedTokenLen	[in] Specifies the size, in bytes, of the encoded time-stamp token.
verifyModes	[in] A bitmask identifying how to verify the Time Attribute Certificate (see below).
signingCertKind	[out] On success, points to a value identifying how the Time Attribute Certificate was verified (see below).

The supported values for verifyModes are:

TTI_VerifyMode_ESSCertID	Checks the SHA-1 hashes in the ESSCertID list.
TTI_VerifyMode_ESSCertIDv2	Checks the SHA-2 hashes in the ESSCertIDv2 list.

`TTI_VerifyMode_ESSCertID | TTI_VerifyMode_ESSCertIDv2`	Checks the SHA-2 hashes in the ESSCertIDv2 list. If no SHA-2 hashes are present, checks the SHA-1 hashes in the ESSCertID list.

The supported values for `signingCertKind` are:

`TTI_Verified_SigningCert_None`	The Time Attribute Certificate was not verified against any hash in either of the ESSCertID or ESSCertIDv2 lists.
`TTI_Verified_SigningCert_ESSCertID`	The Time Attribute Certificate was verified using the SHA-1 hash in the ESSCertID list.
`TTI_Verified_SigningCert_ESSCertIDv2`	The Time Attribute Certificate was verified using the SHA-2 hash in the ESSCertIDv2 list.

3.3.11.2. Return values

If this function succeeds, the return value is zero (`TTI_SUCCESS`).

If this function fails, the return value is a nonzero error code.

`TTI_INVALID_PARAMETER`	`certBuf` and `encodedToken` must not be NULL, and `verifyModes` must be a supported combination of `TTI_VerificationModes` values.
`TTI_ASN_ERROR`	Received unexpected results from an ASN function.
`TTI_TSR_ASN_ERROR`	An unexpected ASN error occurred. To get the ASN error code, call `TTI_GetLastAsnError`.
`TTI_ESSCERTID_FAILED`	The hash of the Time Attribute certificate did not match a value stored in the signed attributes of the time-stamp token.

3.3.11.3. Remarks

This function verifies that the time-stamp token was issued under the Time Attribute certificate. This function is especially useful when the time-stamp token does not contain a Time Attribute certificate and the user wants to verify that the time-stamp token was issued under a TAC that the user retrieved from a previous time-stamp or some other mechanism. This verification is done based on the `verifyModes` bitmask described above. If the Time Attribute Certificate in the encoded time-stamp token is embedded within the set of SignerAttributes and not the certificate list, the TAC will not be verified using ESSCertID or ESSCertIDv2 hashes. This will be indicated by the value `TTI_Verified_SigningCert_None` being set in `signingCertKind`. The cryptographic binding of a time-stamp to a Time Attribute certificate is accomplished by including the appropriate hash of the Time Attribute certificate in ESSCertID or ESSCertIDv2. Both ESSCertID and ESSCertIDv2 are protected by the signature of the time-stamp token. Checking ESSCertIDv2 rather than ESSCertID will succeed only if the TSS is configured to include ESSCertIDv2 in the certificate as per RFC 5816.

3.3.12. TTI_GetTAC_CertInfoEx

Decodes an encoded Time Attribute certificate and writes the encapsulated certificate data into a TTI_TAC_CertInfoEx structure.

```
int TTI_GetTAC_CertInfoEx(
    TTI_TAC_CertInfoEx *    pTAC,
    const byte *            certBuf,
    size_t                  certBufLen );
```

3.3.12.1. Parameters

pTAC	[out] Points to a TTI_TAC_CertInfoEx structure that receives information from the encoded Time Attribute certificate. The TTI_TAC_CertInfoEx structure supports nonce and serial numbers up to 40 bytes.
certBuf	[in] Points to a buffer than contains an encoded Time Attribute certificate.
certBufLen	[in] Specifies the size, in bytes, of the encoded Time Attribute certificate.

3.3.12.2. Return values

If this function succeeds, the return value is zero (TTI_SUCCESS).

If this function fails, the return value is a nonzero error code.

TTI_INVALID_PARAMETER	pTimingMetrics and certBuf must not be NULL.
TTI_ASN_ERROR	Received unexpected results from an ASN function.
TTI_TSR_ASN_ERROR	An unexpected ASN error occurred. To get the ASN error code, call TTI_GetLastAsnError.

3.3.13. Signature validation function

The following section explains the signature validation function.

3.3.13.1. TTI_VerifyTST_SignatureEx

Verifies the signature on an encoded time-stamp token.

```
    int TTI_VerifyTST_SignatureEx(
        const byte *            encodedToken,
        size_t                  encodedTokenLen,
        const byte *            tsaCert,
        size_t                  tsaCertLen,
        TTI_VerificationModes  verifyModes,
        TTI_SigningCert_Kind * signingCertKind );
```

3.3.13.2. Parameters

encodedToken	[in] Points to a buffer that contains an encoded time-stamp token.
encodedTokenLen	[in] Specifies the size, in bytes, of encoded time-stamp token.
tsaCert	[in, optional] Points to a buffer that contains the TSA certificate that signed this token. If this parameter is NULL, this function attempts to verify the signature with a certificate included in the time-stamp token.
tsaCertLen	[in] Specifies the size, in bytes, of the encoded certificate.
verifyModes	[in] A bitmask identifying how to verify the signing certificate (see below).
signingCertKind	[out] On success, points to a value identifying how the signing certificate was verified (see below).

The supported values for verifyModes are:

TTI_VerifyMode_ESSCertID	Checks the first SHA-1 hash in the ESSCertID list.
TTI_VerifyMode_ESSCertIDv2	Checks the first SHA-2 hash in the ESSCertIDv2 list.
TTI_VerifyMode_ESSCertID | TTI_VerifyMode_ESSCertIDv2	Checks the first SHA-2 hash in the ESSCertIDv2 list. If no SHA-2 hashes are present, checks the first SHA-1 hash in the ESSCertID list.

The supported values for signingCertKind are:

TTI_Verified_SigningCert_ESSCertID	The signing certificate was verified using the SHA-1 hash in the ESSCertID list.
TTI_Verified_SigningCert_ESSCertIDv2	The signing certificate was verified using the SHA-2 hash in the ESSCertIDv2 list.

3.3.13.3. Return values

If this function succeeds, the return value is zero (TTI_SUCCESS).

If this function fails, the return value is a nonzero error code.

TTI_INVALID_PARAMETER	encodedToken is not allowed to be NULL and verifyModes must be a supported combination of TTI_VerificationModes values..
TTI_INVALID_TST	encodedToken did not contain a valid encoded time-stamp token.
TTI_INVALID_SIGNATURE	The time-stamp token signature was not valid.
TTI_NO_TSACERT_PRESENT	The encoded time-stamp token did not contain the signing certificate. A time-stamp token contains the signing certificate only if the time-stamp request required that the certificate be included.
TTI_ESSCERTID_FAILED	The hash of the Time Attribute certificate did not match a value stored in the signed attributes of the time-stamp token.

3.3.13.4. Remarks

This function is provided so that API users can validate the signature of a time-stamp token. However, if you have access to other PKI tools, we recommend you use those to validate the signature. Asking the API to validate its own signature is of limited value. In addition to verifying the signature on the time-stamp token, TTI_VerifyTST_Signature verifies the cryptographic binding of the time-stamp token to a particular identity certificate using either ESSCertID or ESSCertIDv2. The function achieves this using the verifyModes bitmask described above. Checking ESSCertIDv2 rather than ESSCertID will succeed only if the TSS is configured to include ESSCertIDv2 in the certificate as per RFC 5816.

3.3.14. Hash functions

The following sections document hash functions.

3.3.14.1. TTI_SHA1_Init

Creates a SHA-1 hash object and returns a handle that can be used to access the object.

```
TTI_SHA1_HANDLE TTI_SHA1_Init();
```

3.3.14.2. Return values

If this function succeeds, the return value is zero (TTI_SUCCESS).

If this function fails, the return value is a nonzero error code.

3.3.14.3. Remarks

Use the TTI_SHA1_Update function to feed data to the hash object. After a successful call to this function, the returned handle must eventually be released with a call to TTI_SHA1_Final.

3.3.14.4. TTI_SHA1_Update

Used to feed data to a specified hash object. Before calling this function, the TTI_SHA1_Init function must be called to get a handle to a hash object.

```
void TTI_SHA1_Update(
    TTI_SHA1_HANDLE hSHA,
    const void *    pData,
    size_t          cbData );
```

3.3.14.5. Parameters

hSHA	[in] Handle of the SHA-1 hash object.
pData	[in] Points to a buffer containing the data to be added to the SHA-1 hash object.
cbData	[in] Number of bytes of data to be added.

3.3.14.6. Remarks

This function may be called multiple times to compute the hash of long or discontiguous data streams.

3.3.14.7. TTI_SHA1_Final

Used to retrieve the value from a hash object and to release the hash object.

```
void TTI_SHA1_Final(
    TTI_SHA1_HANDLE    hSHA,
    byte *             pbHash );
```

3.3.14.8. Parameters

hSHA	[in] Handle of the SHA-1 hash object.
pbHash	[out] Points to a buffer that receives the hash. The buffer must be at least 20 bytes in length.

3.3.14.9. Remarks

After this function is called, the `TTI_SHA1_HANDLE` that was passed in becomes invalid. It must not be used for future calls to `TTI_SHA1_Update` or `TTI_SHA1_Final`.

3.3.14.10. TTI_MD5_Init

Creates an MD5 hash object and returns a handle that can be used to access the object.

```
TTI_MD5_HANDLE TTI_MD5_Init();
```

3.3.14.11. Return values

If this function succeeds, the return value is a nonzero handle. If this function fails, the return value is zero.

3.3.14.12. Remarks

Use the `TTI_MD5_Update` function to feed data to the hash object. After a successful call to this function, the returned handle must eventually be released with a call to `TTI_MD5_Final`.

3.3.14.13. TTI_MD5_Update

Used to feed data to a specified hash object. Before calling this function, the `TTI_MD5_Init` function must be called to get a handle to a hash object.

```
void TTI_MD5_Update(
    TTI_MD5_HANDLE    hMD5,
    const void *      pData,
    size_t            cbData   );
```

3.3.14.14. Parameters

`hMD5`	[in] Handle of the MD5 hash object.
`pData`	[in] Points to a buffer containing the data to be added to the MD5 hash object.
`cbData`	[in] Number of bytes of data to be added.

3.3.14.15. Remarks

This function may be called multiple times to compute the hash of long or discontiguous data streams.

3.3.14.16. TTI_MD5_Final

Used to retrieve the value from a hash object and to release the hash object.

```
    void TTI_MD5_Final(
        TTI_MD5_HANDLE    hMD5,
        byte *            pbHash );
```

3.3.14.17. Parameters

hMD5	[in] Handle of the MD5 hash object.
pbHash	[out] Points to a buffer that receives the hash. The buffer must be at least 20 bytes in length.

3.3.14.18. Remarks

After this function is called, the `TTI_MD5_HANDLE` that was passed in becomes invalid. It must not be used for future calls to `TTI_MD5_Update` or `TTI_MD5_Final`.

3.3.14.19. TTI_SHA256_Init

Creates an `SHA256` hash object and returns a handle that can be used to access the object.

```
    TTI_SHA256_HANDLE TTI_SHA256_Init();
```

3.3.14.20. Return values

If this function succeeds, the return value is a nonzero handle. If this function fails, the return value is zero.

3.3.14.21. Remarks

Use the `TTI_SHA256_Update` function to feed data to the hash object. After a successful call to this function, the returned handle must eventually be released with a call to `TTI_SHA256_Final`.

3.3.14.22. TTI_SHA256_Update

Used to feed data to a specified hash object. Before calling this function, the `TTI_SHA256_Init` function must be called to get a handle to a hash object.

```
    void TTI_SHA256_Update(
        TTI_SHA256_HANDLE    hSHA,
        const void *         pData,
        size_t               cbData  );
```

3.3.14.23. Parameters

`hSHA`	[in] Handle of the SHA hash object.
`pData`	[in] Points to a buffer containing the data to be added to the SHA hash object.
`cbData`	[in] Number of bytes of data to be added.

3.3.14.24. Remarks

This function may be called multiple times to compute the hash of long or discontiguous data streams.

3.3.14.25. TTI_SHA256_Final

Used to retrieve the value from a hash object and to release the hash object.

```
    void TTI_SHA256_Final(
        TTI_SHA256_HANDLE    hSHA,
        byte *               pbHash );
```

3.3.14.26. Parameters

`hSHA`	[in] Handle of the SHA hash object.
`pbHash`	[out] Points to a buffer that receives the hash. The buffer must be at least 20 bytes in length.

3.3.14.27. Remarks

After this function is called, the `TTI_SHA256_HANDLE` that was passed in becomes invalid. It must not be used for future calls to `TTI_SHA256_Update` or `TTI_SHA256_Final`.

3.3.14.28. TTI_SHA384_Init

Creates a SHA384 hash object and returns a handle that can be used to access the object.

```
TTI_SHA384_HANDLE TTI_SHA384_Init();
```

3.3.14.29. Return values

If this function succeeds, the return value is a nonzero handle. If this function fails, the return value is zero.

3.3.14.30. Remarks

Use the TTI_SHA384_Update function to feed data to the hash object. After a successful call to this function, the returned handle must eventually be released with a call to TTI_SHA384_Final.

3.3.14.31. TTI_SHA384_Update

Used to feed data to a specified hash object. Before calling this function, the TTI_SHA384_Init function must be called to get a handle to a hash object.

```
void TTI_SHA384_Update(
    TTI_SHA384_HANDLE    hSHA,
    const void *         pData,
    size_t               cbData   );
```

3.3.14.32. Parameters

hSHA	[in] Handle of the SHA hash object.
pData	[in] Points to a buffer containing the data to be added to the SHA384 hash object.
cbData	[in] Number of bytes of data to be added.

3.3.14.33. Remarks

This function may be called multiple times to compute the hash of long or discontiguous data streams.

3.3.14.34. TTI_SHA384_Final

Used to retrieve the value from a hash object and to release the hash object.

```
void TTI_SHA384_Final(
    TTI_SHA384_HANDLE    hSHA,
    byte *               pbHash );
```

3.3.14.35. Parameters

`hSHA`	[in] Handle of the SHA hash object.
`pbHash`	[out] Points to a buffer that receives the hash. The buffer must be at least 20 bytes in length.

3.3.14.36. Remarks

After this function is called, the `TTI_SHA384_HANDLE` that was passed in becomes invalid. It must not be used for future calls to `TTI_SHA384_Update` or `TTI_SHA384_Final`.

3.3.14.37. TTI_SHA512_Init

Creates a `SHA512` hash object and returns a handle that can be used to access the object.

```
TTI_SHA512_HANDLE TTI_SHA512_Init();
```

3.3.14.38. Return values

If this function succeeds, the return value is a nonzero handle. If this function fails, the return value is zero.

3.3.14.39. Remarks

Use the `TTI_SHA512_Update` function to feed data to the hash object. After a successful call to this function, the returned handle must eventually be released with a call to `TTI_SHA512_Final`.

3.3.14.40. TTI_SHA512_Update

Used to feed data to a specified hash object. Before calling this function, the `TTI_SHA512_Init` function must be called to get a handle to a hash object.

```
void TTI_SHA512_Update(
    TTI_SHA512_HANDLE    hSHA,
    const void *         pData,
    size_t               cbData  );
```

3.3.14.41. Parameters

hSHA	[in] Handle of the SHA hash object.
pData	[in] Points to a buffer containing the data to be added to the SHA512 hash object.
cbData	[in] Number of bytes of data to be added.

3.3.14.42. Remarks

This function may be called multiple times to compute the hash of long or discontiguous data streams.

3.3.14.43. TTI_SHA512_Final

Used to retrieve the value from a hash object and to release the hash object.

```
void TTI_SHA512_Final(
    TTI_SHA512_HANDLE    hSHA,
    byte *               pbHash );
```

3.3.14.44. Parameters

hSHA	[in] Handle of the SHA hash object.
pbHash	[out] Points to a buffer that receives the hash. The buffer must be at least 20 bytes in length.

3.3.14.45. Remarks

After this function is called, the `TTI_SHA512_HANDLE` that was passed in becomes invalid. It must not be used for future calls to `TTI_SHA512_Update` or `TTI_SHA512_Final`.

3.4. Utility functions

The following sections document the utility functions available.

3.4.1. TTI_GetLastAsnError

Used to retrieve extended ASN error codes. This function should be called after an encoding or decoding function returns TTI_TSR_ASN_ERROR.

```
int TTI_GetLastAsnError();
```

3.4.1.1. Return values

Returns the last ASN error code that occurred during an encoding or decoding function. This value is set before an encoding or decoding function returns TTI_TSR_ASN_ERROR.

3.4.1.2. Remarks

There are a large number of error codes that may be returned from this function. These errors usually occur only when an invalid or corrupted buffer is passed to a decode function. Since these errors are unexpected, this document does not contain a complete list of possible values. However, this function provides help with technical support in the case of unexpected errors.

3.4.2. TTI_TSTInfoToIDData

This function will modify the encapsulated content type in a time-stamp token, changing it from id-ct-TSTInfo to id-data.

```
int TTI_TSTInfoToIDData(
    const byte *    encodedToken,
    size_t          encodedTokenLen,
    byte *          encodedObjectBuf,
    size_t *        encodedObjectBufLen  );
```

3.4.2.1. Parameters

encodedToken	[in] Points to a buffer that contains an encoded time-stamp token.
encodedTokenLen	[in] Specifies the size, in bytes, of the encoded time-stamp token.
encodedObjectBuf	[out] Points to a buffer to receive the modified encoded time-stamp token.
encodedObjectBufLen	[in/out] Points to a value specifying the size, in bytes, of the encodedObjectBuf buffer. When the function returns, this value contains the size, in bytes, of the modified encoded time-stamp token.

3.4.2.2. Return values

If this function succeeds, the return value is zero (TTI_SUCCESS).

If this function fails, the return value is a nonzero error code.

TTI_INVALID_PARAMETER	pbEncodedToken and pcbEncodedObjectLength must not be NULL.
TTI_BUFFER_TOO_SMALL	The size indicated by encodedObjectBufLen is too small. When the function returns, the required size is returned in the value pointed to by encodedObjectBufLen.
TTI_ASN_ERROR	Received unexpected results from an ASN function.
TTI_TSR_ASN_ERROR	An unexpected ASN error occurred. To get the ASN error code, call TTI_GetLastAsnError.

3.4.2.3. Remarks

The TSS issues time-stamp tokens that follow the requirements specified in RFC 3161. The encoded time-stamp tokens are CMS SignedData objects with an encapsulated content type of id-ct-TSTInfo.

However, testing has shown that several PKI tool sets cannot handle this embedded content type. Therefore, these tools cannot validate the signature on time-stamp tokens. This TTI_TSTInfoToIDData function provides a temporary solution for using one of these PKI tools. Modifying the encapsulated content type to id-data has no adverse affect on signature validation and allows a time-stamp token signature to be validated by popular tool sets. Now that RFC 3161 exists and the time-stamp protocol is no longer just an Internet Engineering Task Force (IETF) draft, we expect that the popular PKI tool sets will be updated to handle the id-ct-TSTInfo object identifier.

3.4.3. TTI_RemoveAttrCerts

Removes attribute certificates from a time-stamp token.

```
int TTI_RemoveAttrCerts(
    const byte *    encodedToken,
    size_t          encodedTokenLen,
    byte *          encodedObjectBuf,
    size_t *        encodedObjectBufLen );
```

3.4.3.1. Parameters

encodedToken	[in] Points to a buffer that contains an encoded time-stamp token.
encodedTokenLen	[in] Specifies the size, in bytes, of the encoded time-stamp token.

encodedObjectBuf	[out] Points to a buffer to receive the modified encoded time-stamp token.
encodedObjectBufLen	[in/out] Points to a value specifying the size, in bytes, of the encodedObjectBuf buffer. When the function returns, this value contains the size, in bytes, of the modified encoded time-stamp token.

3.4.3.2. Return values

If this function succeeds, the return value is zero (TTI_SUCCESS).

If this function fails, the return value is a nonzero error code.

TTI_INVALID_PARAMETER	pbEncodedToken and pcbEncodedObjectLength must not be NULL.
TTI_BUFFER_TOO_SMALL	The size indicated by encodedObjectBufLen is too small. When the function returns, the required size is returned in the value pointed to by encodedObjectBufLen.
TTI_ASN_ERROR	Received unexpected results from an ASN function.
TTI_TSR_ASN_ERROR	An unexpected ASN error occurred. To get the ASN error code, call TTI_GetLastAsnError.

3.4.3.3. Remarks

If the time-stamp token contains a certificate list, the certificate list will contain at least one attribute certificate, most likely the Time Attribute certificate. Many PKI tool sets cannot handle SignedData objects that contain attribute certificates. Therefore PKI tools cannot validate signatures on time-stamp tokens. The TTI_RemoveAttrCerts function provides a work-around for anyone using one of these PKI tools. Removing the attribute certificates has no adverse affect on signature validation and allows a time-stamp token signature to be validated by popular tool sets.

4. Java API functions and specifications

This section provides an overview of the contents and use of the API/JDK. Detailed documentation of the API Java classes is provided in HTML format in the `docs` subdirectory of your install directory.

4.1. Components

The API/JDK consists of a `tti.jar` file, Javadoc documentation and a sample application that demonstrates proper usage.

4.2. Functional overview

The API/JDK can be used to obtain a time-stamp from a TSS with four basic steps:

- Create an encoded request.
- Submit it to the TSS.
- Decode the result.
- Verify the integrity of the time-stamp.

To obtain a time-stamp from a TSS:

1. Using standard `java.security` classes, generate a digest of the data to be time-stamped.
2. Generate a nonce for the request (a large random number that protects the request against replay attacks.)
3. Create a `TimeStampRequest` object with the digest, nonce, and other relevant information.
4. Call `TimeStampRequest.encodeRequest` to create an ASN.1 encoded version of the request.
5. Create a `TimeStampServerTCP` object with the IP address of a time-stamp server.
6. Call `TimeStampServerTCP.submitRequest` to request and receive the encoded time-stamp token. The encoded time-stamp token is a PKCS #7 SignedData object and the signature can be verified with any cryptographic tool kit that supports PKCS #7.
7. Create a `TimeStampToken` object with the encoded time-stamp token.
8. Call `TimeStampToken.getTSTInfo` to obtain a `TSTInfo` object that contains the time-stamp specific information (such as the time).
9. Verify the integrity of the time-stamp token by checking that the time contained in the

time-stamp is reasonably close to the current system time. Each of these steps is illustrated in the example program `TtiTest.java` included with the API/JDK.

5. Deprecated functions

This appendix provides information about the deprecated API functions and specifications.

> Although the deprecated functions and structures listed in this appendix are operational, they do not support the use of SHA-384 or SHA-512. The hash value size in these structures is limited to 40 bytes, which is not sufficient to support SHA-384 or SHA-512.

5.1. TTI_EncodeTSQ

> This function has been replaced by TTI_EncodeTSQ_Ex. See TTI_EncodeTSQ_Ex for more information. Uses the information in a TTI_TSQ structure to create an encoded time-stamp request.

```
int TTI_EncodeTSQ(
    const TTI_TSQ *         pTSQ,
    byte *                  encodedReq,
    size_t *                encodedReqLen,
    TTI_TransportFormat     transportFormat;
```

5.1.1. Parameters

pTSQ	[in] Points to a populated TTI_TSQ structure. The information in this structure is used to create the encoded time-stamp request.
encodedReq	[out] Points to a buffer to receive the encoded time-stamp request.
encodedReqLen	[in/out] Points to a value specifying the size, in bytes, of the encodedReq buffer. When the function returns, this value contains the size, in bytes, of the encoded time-stamp request.
transportFormat	[in] A flag indicating which type of additional transport encoding should be included in the request.

Currently defined format types are:

TTI_RAW	Returns the encoded time-stamp request with no headers and no special encoding.
TTI_TCP	Returns the encoded time-stamp request prepended with a five-byte TCP header.
TTI_HTTP	Returns the time-stamp request encoded as an HTTP encoded request.
TTI_SMTP	Reserved for future use.

5.1.2. Return values

If this function succeeds, the return value is zero (TTI_SUCCESS).

If this function fails, the return value is a nonzero error code.

TTI_INVALID_PARAMETER	pTSQ and encodedReqLen must not be NULL. transportFormat must be a valid TTI_TransportFormat value.
TTI_BUFFER_TOO_SMALL	The size indicated by encodedReqLen is too small. When the function returns, the required size is returned in the value pointed to by encodedReqLen.
TTI_TSR_ASN_ERROR	An unexpected ASN error occurred. To get the ASN error code, call TTI_GetLastAsnError.
TTI_NOT_SUPPORTED	TTI_SMTP are not supported at this time.

5.1.3. Remarks

This function creates an encoded time-stamp request that includes the information supplied in the TTI_TSQ structure. The request is formatted according to version one of the PKIX Time-Stamp protocol. This function will also encode the request for a particular transport mechanism. At this time, two transport format options are supported: TTI_RAW and TTI_TCP. TTI_RAW returns the encoded time-stamp request with no headers and no special encoding. TTI_TCP returns the request with a five-byte TCP header prepended. This header includes the size of the request and a flag byte set to zero (tsaMsg).

When this function is used with the transportFormat set to TTI_TCP, the resulting encoded time-stamp request may be submitted directly to a TSS via a TCP socket connected to port 318 of the server.

5.2. TTI_DecodeTSQ

This function has been replaced by TTI_DecodeTSQ_Ex. See TTI_DecodeTSQ_Ex for more information. Decodes an encoded time-stamp request and writes the information into a TTI_TSQ structure.

```
int TTI_DecodeTSQ(
    TTI_TSQ *       pTSQ,
    const byte *    encodedReq,
    size_t          encodedReqLen );
```

5.2.1. Parameters

pTSQ	[out] Points to a TTI_TSQ structure that receives information from the encoded time-stamp request.
encodedReq	[in] Points to a buffer that contains an encoded time-stamp request.
encodedReqLen	[in] Specifies the size, in bytes, of the encoded time-stamp request.

5.2.2. Return values

If this function succeeds, the return value is zero (TTI_SUCCESS).

If this function fails, the return value is a nonzero error code.

TTI_INVALID_PARAMETER	pTSQ and encodedReq must not be NULL.
TTI_ASN_ERROR	Received unexpected results from an ASN function.
TTI_TSR_ASN_ERROR	An unexpected ASN error occurred. To get the ASN error code, call TTI_GetLastAsnError.

5.2.3. Remarks

This function can be used to decode a time-stamp request that was encoded with TTI_EncodeTSQ. It is potentially useful if the original TTI_TSQ structure that was used to create the encoded request is not available.

5.3. TTI_GetTST_TSTInfo

> ℹ This function has been replaced by TTI_GetTST_TSTInfoEx. See TTI_GetTST_TSTInfoEx for more information. Decodes an encoded time-stamp token and writes the encapsulated TSTInfo data into a TTI_TSTInfo structure.

```
int TTI_GetTST_TSTInfo(
    TTI_TSTInfo *    pTSTInfo,
    const byte *     encodedToken,
    size_t           encodedTokenLen );
```

5.3.1. Parameters

pTSTInfo	[out] Points to a TTI_TSTInfo structure that receives information from the encoded time-stamp token.

encodedToken	[in] Points to a buffer that contains an encoded time-stamp token.
encodedTokenLen	[in] Specifies the size, in bytes, of the encoded time-stamp token.

5.3.2. Return values

If this function succeeds, the return value is zero (TTI_SUCCESS).

If this function fails, the return value is a nonzero error code.

TTI_INVALID_PARAMETER	encodedResp and tokenBufLen must not be NULL.
TTI_ASN_ERROR	Received unexpected results from an ASN function.
TTI_TSR_ASN_ERROR	An unexpected ASN error occurred. To get the ASN error code, call TTI_GetLastAsnError.

5.3.3. Remarks

This is the core data of a time-stamp token. The TSTInfo is part of the signed data of the time-stamp token and therefore is protected against modification. This function reads and decodes this portion of the time-stamp token and writes the information into a TTI_TSTInfo structure.

5.4. TTI_GetTAC_CertInfo

ℹ This function has been replaced by TTI_GetTAC_CertInfoEx. See TTI_GetTAC_CertInfoEx for more information. Decodes an encoded Time Attribute certificate and writes the encapsulated certificate data into a TTI_TAC_CertInfo structure.

```
int TTI_GetTAC_CertInfo(
    TTI_TAC_CertInfo *    pTAC,
    const byte *          certBuf,
    size_t                certBufLen );
```

5.4.1. Parameters

pTAC	[out] Points to a TTI_TAC_CertInfo structure that receives information from the encoded Time Attribute certificate.
certBuf	[in] Points to a buffer than contains an encoded Time Attribute certificate.

certBufLen	[in] Specifies the size, in bytes, of the encoded Time Attribute certificate.

5.4.2. Return values

If this function succeeds, the return value is zero (TTI_SUCCESS).

If this function fails, the return value is a nonzero error code.

TTI_INVALID_PARAMETER	pTimingMetrics and certBuf must not be NULL.
TTI_ASN_ERROR	Received unexpected results from an ASN function.
TTI_TSR_ASN_ERROR	An unexpected ASN error occurred. To get the ASN error code, call TTI_GetLastAsnError.

5.5. TTI_GetTAC_TimingMetrics

> This function has been replaced by TTI_GetTAC_TimingMetricsEx. See TTI_GetTAC_TimingMetricsEx for more information. Decodes an encoded Time Attribute certificate and writes the encapsulated TimingMetrics attribute data into a TTI_TimingMetrics structure.

```
int TTI_GetTAC_TimingMetrics(
    TTI_TimingMetrics *    pTimingMetrics,
    const byte *           certBuf,
    size_t                 certBufLen );
```

5.5.1. Parameters

pTimingMetrics	[out] Points to a TTI_TimingMetrics structure that receives information from the encoded Time Attribute certificate.
certBuf	[in] Points to a buffer that contains an encoded Time Attribute certificate.
certBufLen	[in] Specifies the size, in bytes, of the encoded Time Attribute certificate.

5.5.2. Return values

If this function succeeds, the return value is zero (TTI_SUCCESS).

If this function fails, the return value is a nonzero error code.

TTI_INVALID_PARAMETER	pTimingMetrics and certBuf must not be NULL.

TTI_ASN_ERROR	Received unexpected results from an ASN function.
TTI_TSR_ASN_ERROR	An unexpected ASN error occurred. To get the ASN error code, call TTI_GetLastAsnError.

5.6. TTI_GetTST_TSACert

> This function has been replaced by TTI_GetTST_TSACertEx. See TTI_GetTST_TSACertEx for more information. Retrieves the encoded TSA certificate from an encoded time-stamp token.

```
int TTI_GetTST_TSACert(
    byte *         certBuf,
    size_t *       certBufLen,
    const byte *   encodedToken,
    size_t         encodedTokenLen );
```

5.6.1. Parameters

certBuf	[out] Points to a buffer that receives the encoded TSA certificate.
certBufLen	[in/out] Points to a value specifying the size, in bytes, of the certBuf buffer. When the function returns, this value contains the size, in bytes, of the encoded TSA certificate.
encodedToken	[in] Points to a buffer that contains an encoded time-stamp token.
encodedTokenLen	[in] Specifies the size, in bytes, of the encoded time-stamp token.

5.6.2. Return Values

If this function succeeds, the return value is zero (TTI_SUCCESS).

If this function fails, the return value is a nonzero error code.

TTI_INVALID_PARAMETER	encodedToken and certBufLen must not be NULL.
TTI_BUFFER_TOO_SMALL	The size indicated by certBufLen is too small. When the function returns, the required size is returned in the value pointed to by certBufLen.
TTI_INVALID_TST	The content of the encodedToken buffer is not recognized as a valid encoded time-stamp token.
TTI_NO_TSACERT_PRESENT	The encoded time-stamp token did not contain the signing certificate. A time-stamp token will only contain the signing certificate if the time-stamp request specified that the certificate be included.

TTI_ESSCERTID_FAILED	The hash of the signing certificate did not match a value stored in the signed attributes of the time-stamp token.

5.6.3. Remarks

If the encoded time-stamp token contains the signing certificate, this function copies the encoded signing certificate into the supplied buffer. This function does not validate the signature on the time-stamp token. By default, this function verifies the signing certificate against the first SHA-2 hash in the ESSCertIDv2 list before the function returns. The ESSCertIDv2 provides cryptographic binding of the time-stamp token to a particular identity certificate, whereas the signature only binds the time-stamp token to the public key. If the TTI_VERIFY_ESSCERTIDV2 environment variable is set to 0, the function will instead carry out the comparable checks against the first SHA-1 hash in the ESSCertID list. Checking ESSCertIDv2 rather than ESSCertID will succeed only if the TSA is configured to include ESSCertIDv2 in the certificate as per RFC 5816.

5.7. TTI_GetTST_TimeAttributeCert

This function has been replaced by `TTI_GetTST_TimeAttributeCertEx`. See TTI_GetTST_TimeAttributeCertEx for more information. Retrieves the encoded Time Attribute certificate from an encoded time-stamp token.

```
int TTI_GetTST_TimeAttributeCert(
    byte *          certBuf,
    size_t *        certBufLen,
    const byte *    encodedToken,
    size_t          encodedTokenLen );
```

5.7.1. Parameters

certBuf	[out] Points to a buffer that receives the encoded Time Attribute Certificate.
certBufLen	[in/out] Points to a value specifying the size, in bytes, of the buffer. When the function returns, this value contains the size, in bytes, of the encoded Time Attribute certificate.
encodedToken	[in] Points to a buffer that contains an encoded time-stamp token.
encodedTokenLen	[in] Specifies the size, in bytes, of the encoded time-stamp token.

5.7.2. Return Values

If this function succeeds, the return value is zero (TTI_SUCCESS).

If this function fails, the return value is a nonzero error code.

TTI_INVALID_PARAMETER	encodedToken and certBufLen must not be NULL.
TTI_BUFFER_TOO_SMALL	The size indicated by certBufLen is too small. When the function returns, the required size is returned in the value pointed to by certBufLen.
TTI_INVALID_TST	The content of the encodedToken buffer is not recognized as a valid encoded time-stamp token.
TTI_NO_TAC_PRESENT	The encoded time-stamp token did not contain the Time Attribute certificate. A time-stamp token will only contain the Time Attribute certificate if the time-stamp request specified that the certificate be included.
TTI_ESSCERTID_FAILED	The hash of the Time Attribute certificate did not match a value stored in the signed attributes of the time-stamp token.

5.7.3. Remarks

If the encoded time-stamp token contains the Time Attribute certificate, this function copies the encoded TAC into the supplied buffer. This function verifies that the timestamp token was issued under the Time Attribute certificate before the function returns. By default, this verification is done by checking the ESSCertIDv2 list of SHA-2 hashes for the SHA-2 hash of the Time Attribute certificate. This check is necessary because the certificate list in a time-stamp token is not protected by the signature. The cryptographic binding of a time-stamp to a Time Attribute certificate is accomplished by including the SHA-2 hash of the Time Attribute certificate in the ESSCertIDv2. The ESSCertIDv2 is protected by the signature of the time-stamp token. If the TTI_VERIFY_ESSCERTIDV2 environment variable is set to 0, the function will instead carry out the comparable operations using SHA-1, checking the ESSCertID list of SHA-1 hashes for the SHA-1 hash of the Time Attribute certificate. Checking ESSCertIDv2 rather than ESSCertID will succeed only if the TSA is configured to include ESSCertIDv2 in the certificate as per RFC 5816.

5.8. TTI_CheckTAC_MatchesTST

This function has been replaced by TTI_CheckTAC_MatchesTSTEx. See TTI_CheckTAC_MatchesTSTEx for more information. Verifies that the time-stamp token was issued under the Time Attribute certificate.

```
int TTI_CheckTAC_MatchesTST(
        const byte *    certBuf,
        size_t          certBufLen,
        const byte *    encodedToken,
        size_t          encodedTokenLen );
```

5.8.1. Parameters

certBuf	[in] Points to a buffer that contains an encoded Time Attribute certificate.
certBufLen	[in] Specifies the size, in bytes, of the encoded Time Attribute certificate.
encodedToken	[in] Points to a buffer that contains an encoded time-stamp token.
encodedTokenLen	[in] Specifies the size, in bytes, of the encoded time-stamp token.

5.8.2. Return Values

If this function succeeds, the return value is zero (TTI_SUCCESS).

If this function fails, the return value is a nonzero error code.

TTI_INVALID_PARAMETER	certBuf and encodedToken must not be NULL.
TTI_ASN_ERROR	Received unexpected results from an ASN function.
TTI_TSR_ASN_ERROR	An unexpected ASN error occurred. To get the ASN error code, call TTI_GetLastAsnError.
TTI_ESSCERTID_FAILED	The hash of the Time Attribute certificate did not match a value stored in the signed attributes of the time-stamp token.

5.8.3. Remarks

This function verifies that the time-stamp token was issued under the Time Attribute certificate. This function is especially useful when the time-stamp token does not contain a Time Attribute certificate and the user wants to verify that the time-stamp token was issued under a TAC that the user retrieved from a previous time-stamp or some other mechanism. By default, this verification is done by checking the ESSCertIDv2 list of SHA-2 hashes for the SHA-2 hash of the Time Attribute certificate. The cryptographic binding of a time-stamp to a Time Attribute certificate is accomplished by including the SHA-2 hash of the Time Attribute certificate in the ESSCertIDv2. The ESSCertIDv2 is protected by the signature on the time-stamp token. If the TTI_VERIFY_ESSCERTIDV2 environment variable is set to 0, the function will instead carry out the comparable checks against SHA-1 hashes in the ESSCertID list. Checking ESSCertIDv2 rather than ESSCertID will succeed only if the TSA is configured to include ESSCertIDv2 in the certificate as per RFC 5816.

5.9. TTI_VerifyTST_Signature

This function has been replaced by TTI_VerifyTST_SignatureEx. See TTI_VerifyTST_SignatureEx for more information. Verifies the signature on an encoded time-stamp token.

```
int TTI_VerifyTST_Signature(
    const byte *    encodedToken,
    size_t          encodedTokenLen,
    const byte *    tsaCert,
    size_t          tsaCertLen );
```

5.9.1. Parameters

encodedToken	[in] Points to a buffer that contains an encoded time-stamp token.
encodedTokenLen	[in] Specifies the size, in bytes, of encoded time-stamp token.
tsaCert	[in, optional] Points to a buffer that contains the TSA certificate that signed this token. If this parameter is NULL, this function attempts to verify the signature with a certificate included in the time-stamp token.
tsaCertLen	[in] Specifies the size, in bytes, of the encoded certificate.

5.9.2. Return Values

If this function succeeds, the return value is zero (TTI_SUCCESS).

If this function fails, the return value is a nonzero error code.

TTI_INVALID_PARAMETER	encodedToken is not allowed to be NULL.
TTI_INVALID_TST	encodedToken did not contain a valid encoded time-stamp token.
TTI_INVALID_SIGNATURE	The time-stamp token signature was not valid.
TTI_NO_TSACERT_PRESENT	The encoded time-stamp token did not contain the signing certificate. A time-stamp token contains the signing certificate only if the time-stamp request required that the certificate be included.
TTI_ESSCERTID_FAILED	The hash of the signing certificate did not match a value stored in the signed attributes of the time-stamp token.

5.9.3. Remarks

This function is provided so that API users can validate the signature of a time-stamp token.

However, if you have access to other PKI tools, we recommend you use those to validate the signature. Asking the API to validate its own signature is of limited value. In addition to verifying the signature on the time-stamp token, `TTI_VerifyTST_Signature` verifies the cryptographic binding of the time-stamp token to a particular identity certificate using either ESSCertID or ESSCertIDv2. By default, the function achieves this by verifying the signing certificate against the first SHA-2 hash in the ESSCertIDv2 list before the function returns. If the TTI_VERIFY_ESSCERTIDV2 environment variable is set to 0, the function will instead carry out the comparable checks against the first SHA-1 hash in the ESSCertID list. Checking ESSCertIDv2 rather than ESSCertID will succeed only if the TSA is configured to include ESSCertIDv2 in the certificate as per RFC 5816.