Tokeneer ID Station
Interface Specification

S.P1229.41.3
Issue: 1.0
Status: Definitive
19th August 2008

Originator

David Painter

Approver

Janet Barnes

Copies to:

NSA

Praxis High Integrity Systems
Project File

SPRE Inc

Contents

1 Introduction

1.1 Background

In order to demonstrate that developing highly secure systems to the level of rigour required by the higher assurance levels of the Common Criteria is possible, the NSA has asked Praxis High Integrity Systems to undertake a research project to re-develop part of an existing secure system (the Tokeneer System) in accordance with their own high-integrity development process. The component of the Tokeneer System that is to be redeveloped is the core functionality of the Token ID Station (TIS). This re-development work will then be used to show the security community that it is possible to develop secure systems rigorously in a cost-effective manner.

1.2 Identification

There are five systems of interest:

- the operational Tokeneer system
- the operational ID Station (a component of the operational Tokeneer system)
- the re-developed ID Station(TIS) (non-operational, but functionally equivalent to the operational ID Station)
- the re-developed TIS core functions (a subset of the software in the re-developed ID Station)
- the re-developed TIS support functions (all of the re-developed ID Station *except* the re-developed ID Station core functions)

This specification defines the interfaces provided by the re-developed TIS support functions. The TIS core functions and the peripheral simulators use these interfaces.

This document defines all the *interfaces* indicated by connectors in Figure 1. Interfaces internal to TIS are implemented as APIs where procedure calls used by the core functions are defined. Interfaces between the support functions and the peripheral simulators take the form of messages transmitted using TCP/IP.

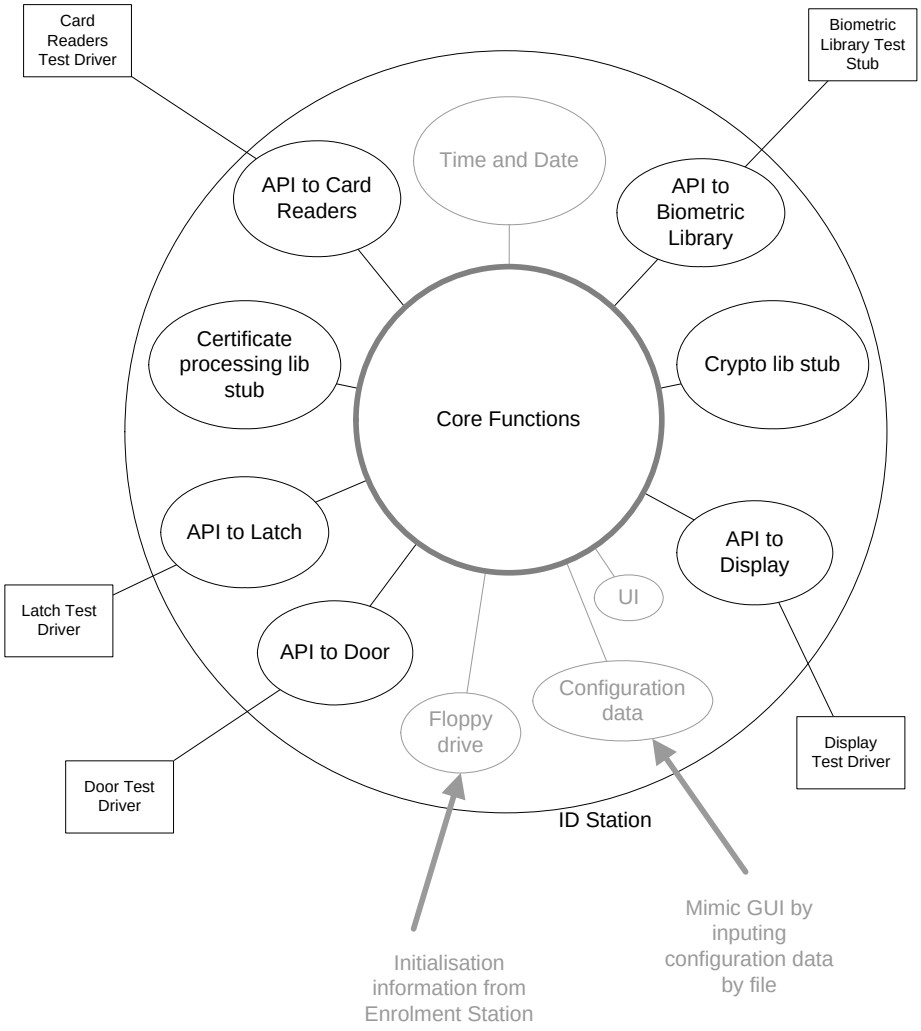

Figure 1: Interfaces with TIS support functions.

1.3 Structure

Section 2 defines the structured content of certificates. The structured content is a model of the actual content of certificates, for example signatures will be used to encode whether they work rather than being true signatures of the data.

Section 3 defines the APIs of each of the TIS support functions provided for use by the TIS core.

Section 4 defines the TCP/IP message content for messages transmitted between the TIS support functions and the simulated peripherals.

Appendix A includes example walkthroughs to illustrate how the system uses the control data embedded in the certificates.

2 Certificate Contents

Each certificate contains various data fields signed by a digital signature. The data fields presented here are intended to be representative of the true certificates used in practice although ASN.1 (Abstract Syntax Notation One) and the associated Basic Encoding Rules (BER) and Distinguished Encoding Rules (DER) typically used in these contexts will not be employed. Structures will be simplified considerably to minimise the functionality that needs to be provided by the Certificate Processing Library. In particular all fields are fixed width so the offset of a particular data item in the certificate is easy to determine.

2.1 Common Data Type Representations

2.1.1 Unsigned32

Used to represent a 32-bit unsigned word, therefore can take values in the range 0..2**32 – 1.

2.1.2 Time

All times will be presented to one-minute accuracy using the following format:

Field	Representation	Size (32-bit words)	Valid values
Year	Unsigned32	1	1990 – 2099
Month	Unsigned32	1	01 – 12
Day	Unsigned32	1	01 – 31
Hour	Unsigned32	1	00 – 23
Minute	Unsigned32	1	00 - 59

It will be assumed that the TIS system is working in the same time frame as appear on the certificates, so time zones will not be recorded.

2.1.3 Validity

Validity periods are constructed from two times. A certificate is valid at any time `t` such that:

```
NotBefore <= t and t <= NotAfter
```

Field	Representation	Size (32-bit words)	Valid values
NotBefore	Time	5	
NotAfter	Time	5	

2.1.4 Name

Names are used to identify the issuer of a certificate and the owner of an ID certificate.

A name will be represented by the following structure; this gives a unique ID for the name and a Text string summary of the name. The text string will not be used for matching.

Field	Representation	Size (32-bit words)	Valid values
ID	Unsigned32	1	Any
Text Length	Unsigned32	1	0 – 39
Text	String40	10	null terminated string of ASCII characters

2.1.5 BaseCertID

The base certificate ID identifies the ID certificate to which an attribute certificate belongs.

This includes the Issuer and Serial Number of the ID certificate.

Field	Representation	Size (32-bit words)	Valid values
Issuer	Name	12	
SerialNumber	Unsigned32	1	any

2.1.6 Role

The role of a token holder is defined as one of the following

User	(00)
Guard	(01)
SecurityOfficer	(02)
AuditManager	(03)

Field	Representation	Size (32-bit words)	Valid values
Role	Unsigned32	1	0 - 3

2.1.7 Class

The class is the ordered classifications on documents, areas and people:

Unmarked	(00)
Unclassified	(01)
Restricted	(02)
Confidential	(03)
Secret	(04)
TopSecret	(05)

Field	Representation	Size (32-bit words)	Valid values
Class	Unsigned32	1	0 – 5

2.1.8 Privilege

The privilege information includes both role and class information. The padding allows future additions to these.

Field	Representation	Size (32-bit words)	Valid values
Role	Role	1	
Class	Class	1	
Padding	Unsigned32 array	2	any

2.1.9 Template

The bio template contains information modelling a fingerprint template; those fields that are explicitly named are extracted by the core TIS for processing. The Template is then passed, in its entirety, through to the Biometrics Library and on to the Fingerprint reader simulator unmodified.

Field	Representation	Size (32-bit words)	Valid values
TemplateLength	Unsigned32	1	1 – 500
RequiredMaxFAR	Integer32	1	any valid RateType (see 3.2.1)
Padding	Unsigned32 array	123	any

2.1.10 AlgorithmIdentifier

This is represented by MechanismType, based on Unsigned32; the valid values should be taken from the crypto library (see section 3.7.1).

2.1.11 PublicKeyInfo

This is partially based on reality and partially fictional. The Algorithm ID is a true field. The remaining data is fictional and replaces the public key data; it provides a numeric identifier for the key rather than the key itself, followed by a key length in bytes. The data type is then padded.

Field	Representation	Size (32-bit words)	Valid values
AlgorithmId	AlgorithmIdentifier	1	
KeyID	Unsigned32	1	Any
KeyLength	Unsigned32	1	0-128
Padding	Unsigned32 array	36	any – data ignored

2.1.12 SignatureData

This represents the signature data that is held in the certificate.

Field	Representation	Size (32-bit words)	Valid values
AlgorithmId	AlgorithmIdentifier	1	
SignatureLength	Unsigned32	1	0 - 128
Signature	Signature	32	

2.1.13 Signature

This is fictional in its content and replaces the encrypted hash, which forms the signature. The remaining data provides the id of the key needed to verify the data and the id of the digest that was signed to create the signature.

Field	Representation	size (32-bit words)	Valid values
KeyID	Unsigned32	1	any
DigestID	Unsigned32	1	any
Padding	Unsigned32 array	30	any – data ignored

2.1.14 DigestInfo

This is a fictional field, used to generate return values from the digest, sign and verify calculations. Note that a digest is calculated prior to signing and verifying certificates. The padding is space reserved for future use.

Field	Representation	Size (32-bit words)	Valid values
DigestID	Unsigned32	1	any, zero will result in a new ID being generated.
SignReturn	Unsigned32	1	appropriate return values for Crypto Library Sign
VerifyReturn	Unsigned32	1	appropriate return values for Crypto Library Verify
Padding	Unsigned32 array	5	any

2.1.15 CryptoControl

This is a fictional field, used to generate return values from the digest calculations:

Field	Representation	Size (32-bit words)	Valid values
DigestUpdateReturn	Unsigned32	1	appropriate return values for Crypto Library DigestUpdate
DigestFinalReturn	Unsigned32	1	appropriate return values for Crypto Library DigestFinal
DigestLength	Unsigned32	1	0 - 32
Digest	DigestInfo	8	

2.2 ID Certificate

This is based on the real ID Certificate. Key differences are:

- in place of a version number there is an encoding of the certificate type; and
- there is an additional field CryptoControlData that drives the crypto library calls with this data.

Field	Representation	Size (32-bit words)	Valid values
CertificateType	Unsigned32	1	0
Serial Number	Unsigned32	1	any
SignatureAlg	AlgorithmIdentifier	1	
Issuer	Name	12	
Validity	Validity	10	
Subject	Name	12	
SubjectPublicKeyInfo	PublicKeyInfo	39	
CryptoControlData	CryptoControl	11	
SignatureData	SignatureData	34	

2.3 Privilege Certificate

This is based on the real Privilege Certificate. Key differences are:

- in place of a version number there is an encoding of the certificate type; and
- there is an additional field CryptoControlData which drives the crypto library calls with this data.

Field	Representation	Size (32-bit words)	Valid values
CertificateType	Unsigned32	1	1
Holder	BaseCertID	13	
Issuer	Name	12	
SignatureAlg	AlgorithmIdentifier	1	
Serial Number	Unsigned32	1	any
AttCertValidity	Validity	10	
Privilege	Privilege	4	
CryptoControl	CryptoControlData	11	
SignatureData	SignatureData	34	

2.4 I&A Certificate

This is based on the real I&A Certificate. Key differences are:

- in place of a version number there is an encoding of the certificate type; and
- there is an additional field CryptoControlData that drives the crypto library calls with this data.

Field	Representation	Size (32-bit words)	Valid values
CertificateType	Unsigned32	1	2
Holder	BaseCertID	13	
Issuer	Name	12	
SignatureAlg	AlgorithmIdentifier	1	
Serial Number	Unsigned32	1	any
AttCertValidity	Validity	10	
Template	Template	125	
CryptoControl	CryptoControlData	11	
SignatureData	SignatureData	34	

2.5 Auth Certificate

This has exactly the same structure as a privilege certificate. The only distinguishing feature is the certificate type value.

Field	Representation	Size (32-bit words)	Valid values
CertificateType	Unsigned32	1	3
Holder	BaseCertID	13	
Issuer	Name	12	
SignatureAlg	AlgorithmIdentifier	1	
Serial Number	Unsigned32	1	any
AttCertValidity	Validity	10	
Privilege	Privilege	4	
CryptoControl	CryptoControlData	11	
SignatureData	SignatureData	34	

3 APIs

3.1 Token Reader Interface

This API mirrors definitions from the PC/SC ICC Resource Manager Definition [1]. Procedures model those given in the Microsoft Security SDK, with one notable exception – the ScardTransmit function. This generic function will be modeled in this system by specific procedures GetIDCert, GetPrivCert, GetIACert, GetAuthCert and UpdateAuthCert. This can be reasoned as abstracting up a level e.g. GetPrivCert is actually a number of Transmit calls, including the container selection, and the multiple reads required to get the certificate. Because of this, and the fact that the returned raw certificate type is being modeled with a fixed length, protocol information (both application-level and transmission-level) need not be modeled. This will minimize the effort required by both Praxis and SPRE to make the interface work. Note that a flag StatusOK has been added to these procedures, to represent the overall status response returned with the data from ScardTransmit.

3.1.1 Types

The following enumerated types are used to represent parameters of the procedures internally to TIS. All outputs from the test drivers to TIS are raw (Unsigned32) to allow erroneous data to be passed back into TIS, which TIS must then deal with. The association between enumerated types and 32 bit values are given at the foot of this section.

ResponseCode type provides a means of examining the success or otherwise of an executed command.

```
type ResponseCode is
   (Success,              -- No error.
    InvalidHandle,        -- Supplied handle is invalid
    InvalidValue,         -- Parameter Value(s) could not be
                          -- properly interpreted
    Cancelled,            -- Action was cancelled by the
                          -- application.
    NoMemory,             -- Not enough memory to complete
                          -- command.
    InsufficientBuffer,   -- Data buffer to receive returned
                          -- data is too small
    UnknownReader,        -- Reader name is not recognized
    Timeout,              -- Timeout has expired
    SharingViolation,     -- ICC cannot be accessed –
                          -- outstanding connections
    NoSmartcard,          -- Required ICC not in device
    UnknownCard,          -- Specified name is not recognized
    ProtoMismatch,        -- Requested protocols incompatible
                          -- with protocol currently in use
```

```
                        -- with the ICC.
      NotReady,         -- Device or card not ready for
                        -- commands
      SystemCancelled,  -- Action cancelled by system
      ReaderUnavailable, -- Device not currently available
                        -- for use
      UnsupportedCard,  -- Reader cannot communicate, due to
                        -- ATR conflicts.
      UnresponsiveCard, -- Card not responding to a reset.
      UnpoweredCard,    -- Power has been removed from the
                        -- card
      ResetCard,        -- Card has been reset, so any
                        -- shared state info is invalid.
      RemovedCard);     -- ICC has been removed.
```

CardStateType reflects the possible states of a card in a reader:

```
   type CardStateType is
      (Absent,       -- No card in the reader
       Present,      -- Card in the reader, but not in position
                     -- for use
       Swallowed,    -- Card in reader, in position for use. Card
                     -- is not powered.
       Powered,      -- Power is being provided to the card, but
                     -- reader driver is unaware of the mode of
                     -- the card
       Negotiable,   -- Card has been reset and is awaiting PTS
                     -- negotiation.
       Specific);    -- Card has been reset and specific
                     -- protocols have been established.
```

ReaderStateType represents the possible states of a reader.

```
   type ReaderStateType is
      (Unaware,     -- State is unknown by the application
       Ignore,      -- Reader should be ignored
       Unavailable, -- Reader is not available for use
       Empty,       -- No card in the reader
       Present,     -- A card is present in the reader
       Mute);       -- An unresponsive card is in the reader
```

String8 is used to represent all possible reader names. These will be null terminated, so actual name length is up to 7 characters.

```
type String8 is String(1..8);
type String8Array is Array(1..10) of String8;
```

ATRType represents the card's answer-to-reset that is sent from the card to the reader after a successful reset. The only part of this that we are interested in is the token's ID, which is used in the validation of certificates held on the card (The ID certificate's serial number is equal to this).

```
type ATRPadding is Array(1..7) of Unsigned32;
type ATRType is record
   TokenID : Unsigned32;
   Padding : ATRPadding;
end record;
```

Certificate types sent and returned to the card are considered to be raw, i.e. a sequence of 32-bit unsigned words. The structure, order and size of the data contained within the certificates is as defined in section 2. GenericRawCert is used to represent any kind of certificate, and contains two fields. CertData is the certificate itself which will be entered at offset 0 and take up as much of the array as necessary; CertLength is the actual amount of data (in bytes) stored in the CertData array.

```
type GenericCertArray is Array(1..250) of Unsigned32;
type GenericRawCert is record
   CertData: GenericCertArray;
   CertLength: Unsigned32;
end record;
```

3.1.2 Initialize

```
procedure Initialize (ResponseCode: out Unsigned32);
```

3.1.2.1 Usage

Initialize is called at TIS initialisation.

3.1.2.2 Behaviour

This allows the application access to the card readers via the resource manager. It sets up the TCP/IP connection (note one connection for the resource manager, not one for each of the readers).

— `ResponseCode` will either indicate success, or will be an error code.

3.1.3 Finalize

```
procedure Finalize (ResponseCode: out Unsigned32);
```

3.1.3.1 Usage

Finalize is called at TIS shut-down.

3.1.3.2 Behaviour

This discontinues communication between TIS and the resource manager, by closing the TCP/IP connection.

— `ResponseCode` will either indicate success, or will be an error code.

3.1.4 ListReaders

```
procedure ListReaders (List: out String8Array;
                       Number: in out Unsigned32;
                       ResponseCode: out Unsigned32);
```

3.1.4.1 Usage

ListReaders will be called at initialisation to provide the TIS with a list of strings used to interface with visible token readers.

— `Number` will be determined by TIS as the number of readers it is expecting to find.

3.1.4.2 Behaviour

Provides a list of known readers.

— `Number` is set to the actual number of readers found.

— `List` will not contain more strings than the expected amount given by the imported '`Number`', so may not contain all readers. In normal operation, would expect "INTREAD" and "EXTREAD" to be returned (in that order), representing the reader internal to the enclave and the reader external to the enclave respectively.

— `ResponseCode` will either indicate success, or will be an error code.

3.1.5 GetStatusChange

```
procedure GetStatusChange (Timeout: in Unsigned32;
                           Reader: in String8;
                           CurrentState: in ReaderStateType;
                           NewState: out Unsigned32;
                           ResponseCode: out Unsigned32);
```

3.1.5.1 Usage

GetStatusChange will be used by TIS as a means of 'polling' the reader, to determine whether a card is present.

— `Timeout` (in milliseconds) will be provided by TIS, and will determine how long to wait for a state change.

— `Reader` will be one of those returned by ListReaders.

— `CurrentState` will be what TIS believes to be the state of the reader;

3.1.5.2 Behaviour

Provides the current state of the named reader.

— `NewState` is set to the actual current state of the reader. This may or may not be the same as `CurrentState`.

— `ResponseCode` will be set to:

- 'TimedOut', if no state change was detected after waiting for `Timeout` milliseconds;
- 'Ok', if a state change was detected, and no other errors occurred; or
- some other error code.

3.1.6 Connect

```
procedure Connect (Reader: in String8;
                   CardHandle: out Unsigned32;
                   ResponseCode: out Unsigned32);
```

3.1.6.1 Usage

Connect is called once GetStatusChange has determined that a card is present in a reader.

— `Reader` will be one of the strings returned by ListReaders.

3.1.6.2 Behaviour

This establishes a connection between TIS and the card in the reader.

— `CardHandle` is set to an identifier that can be used by TIS for communicating with that card. The handle must be calculated in a way that does not allow a card in the 'INTREAD' reader to have the same handle as a card in the 'EXTREAD' reader.

— `ResponseCode` will either indicate success, or will be an error code.

3.1.7 Status

```
procedure Status (CardHandle: in Unsigned32;
                  CState: out Unsigned32;
                  ATR: out ATRType;
                  ResponseCode: out Unsigned32);
```

3.1.7.1 Usage

Status will be called once a card has been connected to provide TIS with the status of the card. This is primarily used to obtain the card's unique ID, contained in the answer-to-reset (ATR). The ATR is only defined if the card is in 'Negotiable' or 'Specific' state, so is ignored when the card is in any other state.

— `CardHandle` will have been returned from the Connect procedure.

3.1.7.2 Behaviour

This provides the status of the referenced card.

— `CState` is set to the current state of the card referenced by `CardHandle`.

— ATR is set to the Answer-To-Reset response if the card is 'Negotiable' or 'Specific'; otherwise it is *don't care*.

— `ResponseCode` will either indicate success, or will be an error code.

3.1.8 Disconnect

```
procedure Disconnect (CardHandle: in Unsigned32;
                      ResponseCode: out Unsigned32);
```

3.1.8.1 Usage

Disconnect is called once TIS has finished communications with the card.

— `CardHandle` will be that introduced by the corresponding Connect call.

3.1.8.2 Behaviour

This terminates the connection opened by a previously called Connect procedure.

— `ResponseCode` will either indicate success, or will be an error code.

3.1.9 GetIDCert

```
procedure GetIDCert (CardHandle: in Unsigned32;
                     RawIDCert: out GenericRawCert;
                     StatusOK: out Boolean;
                     ResponseCode: out Unsigned32);
```

3.1.9.1 Usage

GetIDCert is called by TIS once it is in communication with a card, it is used to obtain the ID certificate from the card.

— `CardHandle` will be that returned from a Connect procedure call, and corresponds to the card currently being processed.

3.1.9.2 Behaviour

Attempts to read the ID certificate from the referenced card.

— `RawIDCert.CertData` will contain the 32-bit words that make up the ID Cert held on the card. The data will start at an offset of 0, and will be ordered as specified in section 2.2.

— `RawIDCert.CertLength` will be set to the actual length of the certificate data (in bytes). Note that, given the definition in section 2.2, the valid length for an ID Cert is 484.

— `StatusOK` represents the overall 'flavour' of the protocol status words returned with the blocks of data. For the purposes of this system, this will either be 'True' (all blocks of data read successfully) or 'False' (e.g. applet failure, memory failure, etc).

— `ResponseCode` will either indicate success, or will be an error code.

3.1.10 GetPrivCert

```
procedure GetPrivCert (CardHandle: in Unsigned32;
                       RawPrivCert: out GenericRawCert;
                       StatusOK: out Boolean;
                       ResponseCode: out Unsigned32);
```

3.1.10.1 Usage

GetPrivCert is called by TIS once it is in communication with a card, it is used to obtain the privilege certificate from the card.

— `CardHandle` will be that returned from a Connect procedure call, and corresponds to the card currently being processed.

3.1.10.2 Behaviour

Attempts to read the Privilege certificate from the referenced card.

— `RawPrivCert.CertData` will contain the 32-bit words that make up the Priv Cert held on the card. The data will start at an offset of 0, and will be ordered as specified in section 2.3.

— `RawPrivCert.CertLength` will be set to the actual length of the certificate data (in bytes). Note that, given the definition in section 2.3, the valid length for an Priv Cert is 348.

— `StatusOK` represents the overall 'flavour' of the protocol status words returned with the blocks of data. For the purposes of this system, this will either be 'True' (all blocks of data read successfully) or 'False' (e.g. applet failure, memory failure, etc).

— `ResponseCode` will either indicate success, or will be an error code.

3.1.11 GetIACert

```
procedure GetIACert (CardHandle: in Unsigned32;
                     RawIACert: out GenericRawCert;
                     StatusOK: out Boolean;
                     ResponseCode: out Unsigned32);
```

3.1.11.1 Usage

GetIACert is called by TIS once it is in communication with a card, it is used to read the I&A certificate from that card.

— `CardHandle` will be that returned from a Connect procedure call, and corresponds to the card currently being processed.

3.1.11.2 Behaviour

Attempts to read the I&A certificate from the referenced card.

— `RawIACert.CertData` will contain the 32-bit words that make up the I&A Cert held on the card. The data will start at an offset of 0, and will be ordered as specified in section 2.4.

— `RawIACert.CertLength` will be set to the actual length of the certificate data (in bytes). Note that, given the definition in section 2.4, the valid length for an I&A Cert is 832.

— `StatusOK` represents the overall 'flavour' of the protocol status words returned with the blocks of data. For the purposes of this system, this will either be 'True' (all blocks of data read successfully) or 'False' (e.g. applet failure, memory failure, etc).

— `ResponseCode` will either indicate success, or will be an error code.

3.1.12 GetAuthCert

```
procedure GetAuthCert (CardHandle: in Unsigned32;
                       RawAuthCert: out GenericRawCert;
                       Exists: out Boolean;
                       StatusOK: out Boolean;
                       ResponseCode: out Unsigned32);
```

3.1.12.1 Usage

GetAuthCert is called by TIS once it is in communication with a card, it is used to read the Auth certificate from that card.

— `CardHandle` will be that returned from a Connect procedure call, and corresponds to the card currently being processed.

3.1.12.2 Behaviour

Attempts to read the Auth certificate from the referenced card. There may not be an Auth Cert on the card.

— `RawAuthCert.CertData` will be:

- the 32-bit words that make up the Auth Cert held on the card, if it exists (the data will start at an offset of 0, and will be ordered as specified in section 2.5); or
- *don't care* if there is not an Auth Cert on the card.

— `RawAuthCert.CertLength` will be:

- set to the actual length of the certificate data (in bytes), if it exists (note that, given the definition in section 2.5, the valid length for an Auth Cert is 348); or
- *don't care* if there is not an Auth Cert on the card.

— `Exists` will be 'True' if there is an Auth Cert on the card, 'False' otherwise.

— `StatusOK` represents the overall 'flavour' of the protocol status words returned with the blocks of data. For the purposes of this system, this will either be 'True' (all blocks of data read successfully) or 'False' (e.g. applet failure, memory failure, etc).

— `ResponseCode` will either indicate success, or will be an error code.

3.1.13 UpdateAuthCert

```
procedure UpdateAuthCert (CardHandle: in Unsigned32;
```

```
                                    RawAuthCert: in GenericRawCert;
                                    StatusOK: out Boolean;
                                    ResponseCode: out Unsigned32);
```

3.1.13.1 Usage

UpdateAuthCert is called if a new Auth Cert has been created for a user

- `CardHandle` will be that returned from a Connect procedure call, and corresponds to the card currently being processed.
- `RawAuthCert` will have been created and signed by TIS.

3.1.13.2 Behaviour

Attempts to write the provided Auth Cert to the user's card.

- `StatusOK` represents the overall 'flavour' of the protocol status words returned by the card. For the purposes of this system, this will either be 'True' (all blocks of data read successfully) or 'False' (e.g. applet failure, memory failure, etc).
- `ResponseCode` will either indicate success, or will be an error code.

3.1.14 Application Inputs

The following numerical representations will be defined for inputs into the application (i.e. those returned by SPRE).

Response codes:

```
0x00000h  Success,
0x00001h  InvalidHandle,
0x00002h  InvalidValue,
0x00003h  Cancelled,
0x00004h  NoMemory,
0x00005h  InsufficientBuffer,
0x00006h  UnknownReader,
0x00007h  TimedOut,
0x00008h  SharingViolation,
0x00009h  NoSmartcard
0x0000Ah  UnknownCard
0x0000Bh  ProtoMismatch,
0x0000Ch  NotReady,
0x0000Dh  SystemCancelled,
0x0000Eh  ReaderUnavailable,
0x0000Fh  UnsupportedCard,
```

```
0x00010h  UnresponsiveCard,
0x00011h  UnpoweredCard,
0x00012h  ResetCard,
0x00013h  RemovedCard.
```

The following card states will be modeled:

```
0x00001h  Absent,
0x00002h  Present,
0x00003h  Swallowed,
0x00004h  Powered,
0x00005h  Negotiable,
0x00006h  Specific.
```

The following reader states will be defined.

```
0x00001h  Unaware,
0x00002h  Ignore,
0x00003h  Unavailable,
0x00004h  Empty,
0x00005h  Present,
0x00006h  Mute.
```

3.2 Biometric Interface

This API has been modeled on the BioAPI specification [2].

3.2.1 Types and Constants

BioApiReturnType models the possible errors that could be raised performing library actions.

```
type BioApiReturnType is Unsigned32;
BioApiOk : constant BioApiReturnType := 0;
```

(A full list of possible error types is given in section 3.1.14.)

RateType is used to model FAR rates. It is a signed 32 bit integer (N) and non-negative values represents a probability of N/(2**31 – 1). –1 indicates the rate is not set. The MaxFAR value used in the Verify function will be the default value, unless there is a valid FAR rate in the individual's Biometric Template.

```
type RateType is Integer range –(2**31)..(2**31 – 1)
RateNotSet : constant RateType := -1;

DefaultMaxFAR : constant RateType := defaultFAR;
```

TemplateType represents the biometric template extracted from the I&A certificate. It will be modeled as a 125 32-bit word array (to represent the max space required by the current Identicator format). The first four bytes give the actual length *n* in bytes of the biometric data, including these four bytes; the next four bytes give the MaxFAR to be used in a match. The next *n–8* bytes is the rest of the biometric data (The core does not actually need to know this, as it simply passes the whole template, as well as the extracted length and MaxFAR criterion, to the Bio library via the Verify procedure).

```
type TemplateType is array (1..125) of Unsigned32;
```

3.2.2 Initialize

```
procedure Initialize (BioReturn: out BioApiReturnType);
```

3.2.2.1 Usage

Initialize is called at TIS initialisation.

3.2.2.2 Behaviour

This is intended to model the Initialisation of the library, and the loading and attaching of the BSP module, as detailed in [2]. It allows the application access to the fingerprint reader via the library. It sets up the TCP/IP connection for this driver.

— A `BioReturn` other than 0 represents an error.

3.2.3 Finalize

```
procedure Finalize (BioReturn: out BioApiReturnType);
```

3.2.3.1 Usage

Finalize is called at TIS shut-down.

3.2.3.2 Behaviour

This is intended to model the Termination of the library, including the detaching and unloading of the BSP module, as detailed in [2]. It discontinues communication between the application and the fingerprint reader via the library, by closing the TCP/IP connection.

— A `BioReturn` other than 0 represents an error.

3.2.4 SamplePresent

```
function SamplePresent return Boolean;
```

3.2.4.1 Usage

SamplePresent is used as a means of 'polling' the reader, to determine whether there is a finger present to sample, and is called prior to attempting a Verify.

3.2.4.2 Behaviour

Returns 'True' if a finger is present, 'False' otherwise.

3.2.5 Verify

```
procedure Verify (Template: in TemplateType;
                  TemplateLength: in Unsigned32;
                  MaxFAR: in RateType;
                  Matched: out Boolean;
                  FARAchieved: out RateType;
                  BioReturn: out BioApiReturnType);
```

3.2.5.1 Usage

Verify is called by TIS when it is ready to check the user's fingerprint against the template held on the user's card.

— `Template` is the unmodified biometric template extracted from the I&A certificate by TIS.

— `TemplateLength` will have been extracted from the fictional field in Template.

— `MaxFAR` will be:

- `DefaultMaxFAR`, if the fictional RequiredMaxFAR field in the Template is `–1`; or
- RequiredMaxFAR otherwise.

3.2.5.2 Behaviour

This is intended to model the BioAPI_Verify function, as detailed in [2].

Verify captures the latest livescan data from the fingerprint reader, and compares it against `Template`. The match is considered a success if the FAR achieved is better or equal to the imported `MaxFAR`.

— `Matched` is set to 'True' if a match was made given the `MaxFAR` criterion, 'False' otherwise.

— `FARAchieved` indicates the closeness of the match. This is returned even if `Matched` is 'False'.

— A `BioReturn` other than 0 represents an error.

3.2.6 Application Inputs

Error codes have been modeled to mirror those introduced in [2].

3.2.6.1 High Level framework errors:

```
(0x0001)   InternalError
           -- Internal error at library level
(0x0002)   MemoryError
           -- Memory error at library level
(0x000A)   FunctionFailed
           -- Function failed for an unknown reason
(0x0046)   InvalidData
           -- Data in an input parameter is invalid
(0x0102)   BioApiNotInitialized
           -- A function is called prior to initializing the
           -- API.
(0x0116)   ModuleLoadFailed
           -- BSP Module Load function failed
(0x0118)   ModuleUnloadFailed
           -- BSP Module Load function failed
```

3.2.6.2 BSP Level errors:

```
(0x1001)   BspInternalError
```

```
                -- Internal error at BSP level
        (0x1002)  BspMemoryError
                -- Memory error at BSP level
        (0x100A)  BspFunctionFailed
                -- Function failed for unknown reason
        (0x1046)  BspInvalidData
                -- Data in an input parameter is invalid
        (0x1101)  BspUnableToCapture
                -- Unable to capture raw samples from the device
        (0x1103)  BspTimeoutExpired
                -- Function terminated due to timeout
        (0x1105)  BspBirSignatureFailure
                -- Unable to validate signature on the BIR
        (0x110D)  BspInconsistentPurpose
                -- Purpose recorded in the BIR, and the requested
                -- purpose, are inconsistent with the function
                -- being performed.
```

3.2.6.3 Device Level Errors:

```
        (0x2001)  DeviceLevelError
                -- A generic device level error
```

3.3 Door Interface

3.3.1 Types

Enumerated type to model the state of the Door.

```
type DoorState is (Uninitialized, Error, Open, Closed);
```

Open means the door does not prevent a human from entering or leaving the enclave.
Closed means the door prevents a human from entering or leaving the enclave. To enter or leave, the door must first be *opened*.
Uninitialized is the state prior to the Initialize function being called.
Error is the state when the Initialize function has been called, but the 'expected' state cannot be determined (Open/Closed).

The door state returned by SPRE is raw (Unsigned32), to allow erroneous data to be returned. The numerical representation of DoorState is:

```
Uninitialized => 1
Error => 2
Open => 3
Closed => 4
```

3.3.2 InitializeDoor

```
procedure InitializeDoor(Success: out Boolean);
```

3.3.2.1 Usage

This procedure is called at TIS initialisation.

3.3.2.2 Behaviour

It gains access to the door test driver, by setting up the TCP/IP connection.

— `Success` indicates whether the action was performed successfully.

3.3.3 FinalizeDoor

```
procedure FinalizeDoor(Success: out Boolean);
```

3.3.3.1 Usage

This procedure is called to relinquish access to the door test driver.

3.3.3.2 Behaviour

It closes the TCP/IP connection.

— `Success` indicates whether the action was performed successfully.

3.3.4 GetDoorState

```
function GetDoorState return Unsigned32;
```

3.3.4.1 Usage

GetDoorState is used by TIS as a means of 'polling' the door to determine the state of the door.

3.3.4.2 Behaviour

Should return the current state of the door, a value of the enumeration type DoorState.

3.4 Latch Interface

3.4.1 InitializeLatch

```
procedure InitializeLatch(Success: out Boolean);
```

3.4.1.1 Usage

This procedure is called at TIS initialisation

3.4.1.2 Behaviour

It gains access to the latch test driver, by setting up the TCP/IP connection.

— `Success` indicates whether the action was performed successfully.

3.4.2 FinalizeLatch

```
procedure FinalizeLatch(Success: out Boolean);
```

3.4.2.1 Usage

This procedure is called to relinquish access to the latch test driver.

3.4.2.2 Behaviour

It closes the TCP/IP connection.

— `Success` indicates whether the action was performed successfully.

3.4.3 Unlock

```
procedure Unlock;
```

3.4.3.1 Usage

Called by TIS when a user has been granted access to the enclave.

3.4.3.2 Behaviour

Sends a request to unlock the latch immediately. Note that the Unlock procedure does not attempt to lock the latch after a timeout period. We are intending to build an abstraction layer to implement this behaviour internally.

3.4.4 Lock

```
procedure Lock;
```

3.4.4.1 Usage

Called by TIS to lock the latch immediately to return the enclave to a secure state.

3.4.4.2 Behaviour

Sends a request to lock the latch immediately.

3.5 Alarm Interface

3.5.1 Initialize

```
procedure Initialize(Success: out Boolean);
```

3.5.1.1 Usage

This procedure is called at TIS initialisation.

3.5.1.2 Behaviour

It gains access to the alarm test driver, by setting up the TCP/IP connection.

— `Success` indicates whether the action was performed successfully.

3.5.2 Finalize

```
procedure Finalize(Success: out Boolean);
```

3.5.2.1 Usage

This procedure is called to relinquish access to the alarm test driver.

3.5.2.2 Behaviour

It closes the TCP/IP connection.

— `Success` indicates whether the action was performed successfully.

3.5.3 Activate/Deactivate

```
procedure Activate;
procedure Deactivate;
```

3.5.3.1 Usage

These procedures are used to switch the alarm between ‘Silent’ and ‘Alarming’

3.5.3.2 Behaviour

Send messages to active and de-active the alarm.

3.6 Display Interface

3.6.1 Types

String50 is used to represent all possible messages to be displayed. Messages will be null terminated, so actual maximum length is 49 characters. It has been assumed that 49 characters is sufficient to display all required messages. The allowed character set is yet to be determined.

```
type String50 is String(1..50);
```

3.6.2 Initialize

```
procedure Initialize(Success: out Boolean);
```

3.6.2.1 Usage

This procedure is called at TIS initialisation.

3.6.2.2 Behaviour

It gains access to the Display test driver, by setting up the TCP/IP connection.

— `Success` indicates whether the action was performed successfully.

3.6.3 Finalize

```
procedure Finalize(Success: out Boolean);
```

3.6.3.1 Usage

This procedure is called to relinquish access to the Display test driver.

3.6.3.2 Behaviour

It closes the TCP/IP connection.

— `Success` indicates whether the action was performed successfully.

3.6.4 GetMaxTextSizeTop/GetMaxTextSizeBottom

```
function GetMaxTextSizeTop return Unsigned32;
function GetMaxTextSizeBottom return Unsigned32;
```

3.6.4.1 Usage

These two functions will be called at TIS initialisation.

3.6.4.2 Behaviour

These determine the size (in characters) of the top and bottom rows of the display.

The lowest size these functions can return is 20. Based on the assumption that a 49 character string is the largest that can be written to a line, we would expect the size returned here to be less than or equal to 49.

3.6.5 SetTopText

```
procedure SetTopText(TopText: in String50;
                     Length: in Unsigned32;
                     Written: out Boolean);
```

3.6.5.1 Usage

SetTopText is called when TIS wishes to display a message on the top line of the display.

— `TopText` will be such that the length of the string contained within it is less than or equal to the size of the display line, returned by GetMaxTextSizeTop.

— `Length` should be equal to the number of printable characters contained in `TopText`.

3.6.5.2 Behaviour

Writes the supplied text to the top line of the display.

— `Written` indicates whether the string was written to the top line.

3.6.6 SetBottomText

```
procedure SetBottomText(BottomText: in String50;
                        Length: in Unsigned32;
                        Written: out Boolean);
```

3.6.6.1 Usage

SetBottomText is called when TIS wishes to display a message on the bottom line of the display.

— `BottomText` will be such that the length of the string contained within it is less than or equal to the size of the display line, returned by GetMaxTextSizeBottom.

— `Length` should be equal to the number of printable characters contained in `BottomText`.

3.6.6.2 Behaviour

Writes the supplied text to the bottom line of the display.

— `Written` indicates whether the string was written to the bottom line.

3.6.7 SetTopTextScrollable

```
procedure SetTopTextScrollable(ScrollText: in String50;
                               Length: in Unsigned32;
                               Written: out Boolean);
```

3.6.7.1 Usage

SetTopTextScrollable is used when a message is too long to be displayed statically on the top and botton lines of the display.

— `ScrollText` should be such that the length of the string contained within it is greater than the size of the display line (returned by GetMaxTextSizeTop), but less than 50 characters.

— `Length` should be equal to the number of printable characters contained in `ScrollText`.

3.6.7.2 Behaviour

Scrolls the supplied text on the top line of the display.

— `Written` indicates whether the string was written to the top line.

3.7 Crypto Library

The Crypto Library is a stub which is controlled by the inputs used in calls to the library.

The procedures used are based very closely on those given in the Cryptoki interface [4]. The library's main tasks are to store the ID station's key pair and the public keys of other entities, and to perform digesting, signing and verifying. As our Crypto facility is most likely to be a 'soft' token, there is a single software slot that we are aware of, and the token is (permanently) inserted. Because of this we only require one session, and we won't need to model the C_GetSlotList and C_GetSlotInfo functions.

Keys are stored as objects (a group of attributes), and are only accessible by a KeyHandle, supplied by the library:

```
KeyHandle → {Owner; KeyID; KeyLength; IsPublic}
```

This data is extracted at enrolment from the ID certificates. The CA ID certificates are tackled first, since these have been self signed, so all required data is stored within the certificate. Once these keys have been extracted, all other key data can be obtained. The only private key that should be stored in the database is that of the TIS itself.

A limitation of the stub is that it will not maintain the database through a power down i.e. enrolment will need to take place each time the TIS is powered up.

3.7.1 Types

As this library will only be storing key objects we are only interested in Key templates. Key data in this system is replaced by dummy keys, consisting of an Owner, a Key ID, a KeyLength, and a Boolean flag indicating whether the key is public or private. KeyTemplate includes these as attributes, and a mask to determine which of the attributes are defined (this is primarily for the Find procedure (see 3.7.5 - 3.7.7), where we may want to find e.g. all public keys). Padding is included to retain a sensible size (128 bytes).

```
type KeyPadding is array(1..67) of Unsigned8;
type KeyTemplate is record
      AttrMask : Unsigned32;     -- 4 bytes
      Owner : NameType;          -- 48 bytes
      KeyID : Unsigned32;        -- 4 bytes
      KeyLength : Unsigned32;    -- 4 bytes
      IsPublic : Boolean;        -- 1 byte
      Padding : KeyPadding;      -- 128 – 61 = 67 bytes
   end record;
```

Each attribute will have a corresponding bit in AttrMask, which will be set if the attribute is defined:

```
OwnerMask : constant Unsigned32 := 1;
KeyIDMask : constant Unsigned32 := 2;
KeyLengthMask : constant Unsigned32 := 4;
```

```
IsPublicMask : constant Unsigned32 := 8;
```

DigestType and SignatureType mirror DigestInfo and Signature defined in section 2.

```
type DigestPadding is array(1..5) of Unsigned32;
type DigestType is record
   DigestID : Unsigned32;
   SignReturn : Unsigned32;
   VerifyReturn : Unsigned32;
   Padding : DigestPaddingType;
end record;

type SignaturePadding is array(1..30) of Unsigned32;
type SignatureType is record
   KeyID : Unsigned32;
   DigestID : Unsigned32;
   Padding : SignaturePaddingType;
end record;
```

MechanismType represents the Mechanism to be used in the crypto operations. The recognised mechanisms have defined numerical values, and are such that combinations of mechanisms are the dot products of the single mechanisms. We are proposing that the library stub support the following mechanisms:

```
type MechanismType is Unsigned32;
-- signing/verifying
RSA           : constant MechanismType := 0x00000001h;
-- digesting
MD2           : constant MechanismType:= 0x00000200h;
MD5           : constant MechanismType:= 0x00000210h;
SHA_1         : constant MechanismType:= 0x00000220h;
RIPEMD128     : constant MechanismType:= 0x00000230h;
RIPEMD160     : constant MechanismType:= 0x00000240h;
-- combined mechanisms…
MD2_RSA       : constant MechanismType:= 0x00000201h;
MD5_RSA       : constant MechanismType:= 0x00000211h;
SHA1_RSA      : constant MechanismType:= 0x00000221h;
RIPEMD128_RSA : constant MechanismType:= 0x00000231h;
RIPEMD160_RSA : constant MechanismType:= 0x00000241h;
```

If any other mechanism is passed to the TIS, the library will return MechanismInvalid when attempting to digest/verify.

100ByteArray is an array of 25 32-bit unsigned words.

```
type 100ByteArray is array(1..25) of Unsigned32;
```

3.7.2 Initialize

```
procedure Initialize(ReturnValue: out Unsigned32);
```

3.7.2.1 Usage

This operation is used to initialize the crypto library at TIS startup.

3.7.2.2 Behaviour

Models C_Initialize and C_OpenSession.

— `ReturnValue` will be:

- 'Ok' if the library has not already been initialized; or
- 'CryptokiAlreadyInitialized' if the library has already been initialized.

3.7.3 Finalize

```
procedure Finalize(ReturnValue: out Unsigned32);
```

3.7.3.1 Usage

This operation is used to finalize the crypto library at TIS startup.

3.7.3.2 Behaviour

Models C_CloseSession and C_Finalize.

— From Finalize, `ReturnValue` will be:

- 'Ok' if the library has been initialized; or
- 'CryptokiNotInitialized' if the library has not been initialized.

3.7.4 CreateObject

```
procedure CreateObject(Template : in KeyTemplate;
                       ObjectHandle : out Unsigned32;
                       ReturnValue : out Unsigned32);
```

3.7.4.1 Usage

CreateObject will be used to store a key.

— `Template` will have been constructed from information held on the enrolment floppy. For example, for public keys:

- `Owner` will be IDCert.Subject;
- `KeyID` will be IDCert.SubjectPublicKeyInfo.KeyID;
- `KeyLength` will be IDCert.SubjectPublicKeyInfo.KeyLength; and
- `IsPublic` will be True.

3.7.4.2 Behaviour

Stores the supplied key.

— `ObjectHandle` will be determined internally, and will be unique to that key.

— `ReturnValue` will always be 'Ok'.

3.7.5 FindObjectsInit

```
procedure FindObjectsInit(Template : in KeyTemplate;
                          ReturnValue : out Unsigned32);
```

3.7.5.1 Usage

The FindObjects set of procedures are used to search the crypto library for an object matching a given template.

— `Template` imported to FindObjectsInit will be:

- extracted from the Issuer field of the certificate currently being validated, if a key is required for validation (that is the Owner attribute will be defined); or
- some other template derived by TIS, depending on the information sought (i.e. if ALL public keys are wanted, then the only defined attribute of Template will be IsPublic).

3.7.5.2 Behaviour

Models C_FindObjectsInit.

FindObjectsInit obtains the Template on which to base the search.

— `ReturnValue` will be 'Ok', unless:

- a find is already underway ('OperationActive'); or

- the library is not initialized ('CryptokiNotInitialized').

3.7.6 FindObjects

```
procedure FindObjects(HandleCount : in out Unsigned32;
                      ObjectHandles : out HandleArray;
                      ReturnValue : out Unsigned32);
```

3.7.6.1 Usage

FindObjects is called after FindObjectsInit.

— `HandleCount` as an input is the number of matches we want/are expecting.

3.7.6.2 Behaviour

Models C_FindObjects.

FindObjects continues the search, returning found matches.

— Each handle in `ObjectHandles` will be determined internally, and will be unique to that key. The number of handles returned in `ObjectHandles` will be less than or equal to the imported `HandleCount`.

— `HandleCount` as an output will be the actual number found.

— `ReturnValue` will be 'Ok', unless:

- the find has not been initialised ('OperationNotInitialized'); or
- the library is not initialized ('CryptokiNotInitialized').

3.7.7 FindObjectsFinal

```
procedure FindObjectsFinal(ReturnValue : out Unsigned32);
```

3.7.7.1 Usage

Used to terminate a search.

3.7.7.2 Behaviour

Models C_FindObjectsFinal.

FindObjectsFinal finalizes the find operation.

— `ReturnValue` will be 'Ok', unless:

- the find has not been initialised ('OperationNotInitialized'); or
- the library is not initialized ('CryptokiNotInitialized').

3.7.8 DigestInit

```
procedure DigestInit(Mechanism : in MechanismType;
                     ReturnValue : out Unsigned32);
```

3.7.8.1 Usage

The digest operations will be used when verifying signed certificates, and when the system is signing an authorization certificate.

— `Mechanism` will have been extracted by TIS from the Signature field of the certificate.

3.7.8.2 Behaviour

DigestInit initializes the operation, and models C_DigestInit. The digest mechanism to be used is determined from `Mechanism` using a bit mask.

— `ReturnValue` will be 'Ok', unless:

- The mechanism derived from `Mechanism` is not used for digesting ('MechanismInvalid');
- The mechanism derived from `Mechanism` is not recognized ('MechanismInvalid');
- a digest operation is already in progress ('OperationActive'); or
- the library is not initialized ('CryptokiNotInitialized').

3.7.9 DigestUpdate

```
procedure DigestUpdate(DataBlock : in 100ByteArray;
                       ByteCount : in Unsigned32;
                       ReturnValue : out Unsigned32);
```

3.7.9.1 Usage

DigestUpdate is called a number of times to read all of the raw certificate data, held in a GenericRawCert array. The whole of the certificate must be read in to ensure that the CryptoControl field is visible.

— `DataBlock` is the (100 byte maximum) block of certificate data to be added to the digest.

— `ByteCount` represents the actual number of certificate data bytes held in `DataBlock`, and will be determined from the CertLength field of the raw cert.

3.7.9.2 Behaviour

Models C_DigestUpdate – the activity of performing a digest.

— `ReturnValue` will be 'Ok', unless:

- it is the last call of DigestUpdate (set to the DigestUpdateReturn value in the CryptoControl field of the certificate);
- the digest is not initialized ('OperationNotInitialized'); or
- the library is not initialized ('CryptokiNotInitialized').

3.7.10 DigestFinal

```
procedure DigestFinal(Digest : out DigestType;
                      DigestLength : out Unsigned32;
                      ReturnValue : out Unsigned32);
```

3.7.10.1 Usage

Called after all of the certificate has been read in via DigestUpdate.

3.7.10.2 Behaviour

Models C_DigestFinal. Finalizes the digest operation, and returns the digest.

— `Digest` will be determined by the Cert.CryptoControl.Digest data, as follows:

- `DigestID` – set to DigestID if it is non-zero, otherwise set to the sum of the 32-bit integers (unsigned) that make up the raw certificate data, multiplied by its position in the certificate, modulo 2**32 i.e.

$$\left[\sum_{pos} (pos * RawIDCert.CertData(pos)) \right] \mod (2^{32})$$

where pos ∈ 1 .. RawXXCert.CertLength

- `SignReturn` – set to SignReturn;
- `VerifyReturn` – set to VerifyReturn;
- `Padding` – an array of 5 32-bit words, of value *don't care*.

— `DigestLength` is set to Cert.CryptoControl.DigestLength.

— `ReturnValue` is set to Cert.CryptoControl.DigestFinalReturn, unless:

- the DigestLength is invalid ('DataLenRange');

- not all of the certificate is read in ('FunctionFailed');
- the digest is not initialized ('OperationNotInitialized'); or
- the library is not initialized ('CryptokiNotInitialized').

3.7.11 Sign

```
procedure Sign(Mechanism : in MechanismType;
               KeyHandle : in Unsigned32;
               Digest : in DigestType;
               DigestLength : in Unsigned32;
               Signature : out SignatureType;
               SigLength : out Unsigned32;
               ReturnValue : out Unsigned32);
```

3.7.11.1 Usage

Used when TIS has created an Auth Cert.

— `Mechanism` will be RSA – TIS will always use the RSA mechanism to sign Authorization certificates.

— `KeyHandle` should represent the private key of the ID station, and will be obtained from the crypto library by the find procedures.

— `Digest` and `DigestLength` are those produced from the digest procedures.

3.7.11.2 Behaviour

Models C_SignInit and C_Sign. Sign produces a signature from the given data.

— `Signature` will be:

- `KeyID`, determined by the imported `KeyHandle`.
- `DigestID`, extracted from the imported Digest.
- `Padding` - an array of 30 32-bit words, of value *don't care*.

— `SigLength` will be set to the KeyLength, determined by the imported `KeyHandle`.

— `ReturnValue` will be SignReturn from the imported Digest, unless:

- the library is not initialized ('CryptokiNotInitialized').

3.7.12 Verify

```
procedure Verify(Mechanism : in MechanismType;
                 KeyHandle : in Unsigned32;
                 Digest : in DigestType;
                 DigestLength : in Unsigned32;
                 Signature : in SignatureType;
                 SigLength : in Unsigned32;
                 ReturnValue : out Unsigned32);
```

3.7.12.1 Usage

The Verify procedure is called to perform the verification of the signature appended to a certificate.

— `Mechanism` will be extracted from the Signature field of the Cert being processed.

— `KeyHandle` will have been obtained from the library using the find procedure.

— `Digest` and `DigestLength` will have been produced from the digest procedures, and will represent the digest of the Cert being processed.

— `Signature` and `SigLength` will be taken from the SignatureData of the Cert being processed.

3.7.12.2 Behaviour

Models C_VerifyInit and C_Verify; the process of verifying the signature against the supplied digest.

The verify mechanism to be used is determined from `Mechanism` using a bit mask.

— `ReturnValue` is set to the VerifyReturn field of the Digest, unless:

- The mechanism derived from `Mechanism` is not for verifying ('MechanismInvalid');
- The mechanism derived from `Mechanism` is not recognized ('MechanismInvalid');
- `KeyHandle` does not 'match' the KeyID in the Signature ('SignatureInvalid');
- the DigestIDs contained in `Digest` and `Signature` don't match ('SignatureInvalid');
- `SigLength` is greater than the key length ('SignatureLenRange'); or
- the library is not initialized ('CryptokiNotInitialized').

3.7.13 GetAttributeValue

```
procedure GetAttributeValue(KeyHandle : in Unsigned32;
                            Template : in out KeyTemplate;
```

```
                                        ReturnValue : out Unsigned32);
```

3.7.13.1 Usage

— `KeyHandle` will have been obtained from the library using the find procedure.

— As an input, `Template` indicates the attributes of interest, by having the relevant bits set in the AttrMask field (at least one bit must be set).

3.7.13.2 Behaviour

Attempts to extract attribute data from the object pointed to by `KeyHandle`. Models C_GetAttributeValue.

— As an output, `Template` will be filled in with the attributes of interest, taken from the object pointed to by `KeyHandle`.

— `ReturnValue` will always be 'Ok', since the nature of the database is that all objects have the same template, so all possible attributes will be defined.

3.7.14 Return Values.

The following return values are defined to model those in Cryptoki [4]. Note that not all errors have been mirrored:

```
Ok                          0x00000000
HostMemory                  0x00000002
GeneralError                0x00000005
FunctionFailed              0x00000006
ArgumentsBad                0x00000007
AttributeReadOnly           0x00000010
AttributeTypeInvalid        0x00000012
AttributeValueInvalid       0x00000013
DataInvalid                 0x00000020
DataLenRange                0x00000021
DeviceError                 0x00000030
DeviceMemory                0x00000031
FunctionCanceled            0x00000050
KeyHandleInvalid            0x00000060
KeySizeRange                0x00000062
KeyTypeInconsistent         0x00000063
KeyFunctionNotPermitted     0x00000068
MechanismInvalid            0x00000070
MechanismParamInvalid       0x00000071
ObjectHandleInvalid         0x00000082
```

```
OperationActive                0x00000090
OperationNotInitialized        0x00000091
SignatureInvalid               0x000000C0
SignatureLenRange              0x000000C1
TemplateIncomplete             0x000000D0
TemplateInconsistent           0x000000D1
BufferTooSmall                 0x00000150
CryptokiNotInitialized         0x00000190
CryptokiAlreadyInitialized     0x00000191
```

3.8 Certificate Processing Library

This stub essentially performs two functions – to extract data from 'raw' certificates, and to construct 'raw' authorization certificates.

3.8.1 Types

Type GenericRawCert is defined in 3.1.1. MechanismType, DigestType and SignatureType are defined in 3.7.1.

The XXCertData types represent the actual data stored in the certificate and as such, don't include the signature data or the fictional CryptoControl data. Certificate structures are defined in section 2.

```
type IDCertData is record
   CertType : Unsigned32;
   SerialNumber : Unsigned32;
   SigAlgId : MechanismType;
   Issuer : NameType;
   Validity : ValidityType;
   Subject : NameType;
   SubjectPublicKeyInfo : PublicKeyInfoType;
end record;

type PrivCertData is record
   CertType : Unsigned32;
   Holder : BaseCertID;
   Issuer : NameType;
   SigAlgId : MechanismType;
   SerialNumber : Unsigned32;
   AttCertValidity : ValidityType;
   Privilege : PrivilegeType;
end record;

type AuthCertData is new PrivCertData;

type IACertData is record
   CertType : Unsigned32;
   Holder : BaseCertID;
   Issuer : NameType;
   SigAlgId : MechanismType;
   SerialNumber : Unsigned32;
   AttCertValidity : ValidityType;
```

```
   Template : TemplateType;
end record;
```

SignatureData includes the signature algorithm ID and SignatureLength as well the actual Signature.

```
type SignatureData is record
   AlgorithmID : MechanismType;
   SigLength : Unsigned32;
   Signature: SignatureType;
end record;
```

3.8.2 ExtractXXCertData

```
procedure ExtractIDCertData(RawIDCert : in GenericRawCert;
                            IDCert : out IDCertData;
                            ExtractSuccess : out Boolean);
procedure ExtractPrivCertData(RawPrivCert : in GenericRawCert;
                              PrivCert : out PrivCertData;
                              ExtractSuccess : out Boolean);
procedure ExtractIACertData(RawIACert : in GenericRawCert;
                            IACert : out IACertData;
                            ExtractSuccess : out Boolean);
procedure ExtractAuthCertData(RawAuthCert : in GenericRawCert;
                              AuthCert : out AuthCertData;
                              ExtractSuccess : out Boolean);
```

3.8.2.1 Usage

The Extract procedures are used to take the raw certificate data extracted from the user's token, and convert into the correct (internal) certificate structure.

— `RawXXCert` is the data obtained from the Token reader interface.

3.8.2.2 Behaviour

Extracts the constituent components of a certificate from its raw representation.

— `XXCert` is constructed from `RawXXCert`, using appropriate offsets to find particular fields. The 32-bit words are checked to determine whether the values are in the valid ranges (e.g. CertificateType for an ID certificate is 0), and then inserts them into the certificate record structure.

— `ExtractSuccess` will be:

- 'True' if all the range checks pass; or
- 'False' If one or more of the range checks fail.

3.8.3 ObtainSignatureData

```
procedure ObtainSignatureData(RawCert : in GenericRawCert;
                              SigData : out SignatureData;
                              ObtainSuccess : out Boolean);
```

3.8.3.1 Usage

— `RawCert` is the data obtained from the Token reader interface.

3.8.3.2 Behaviour

This procedure extracts the signature data from the raw certificate.

— `SigData` is extracted from `RawCert` using the CertLength field to determine the offset of the signature data (possible since SignatureData is a fixed size). No range checks are performed on the signature data.

— `ObtainSuccess` will be:

- 'False' If the CertLength field is not as expected; or
- 'True' otherwise.

3.8.4 ConstructAuthCert

```
procedure ConstructAuthCert(AuthCert: in AuthCertData;
                            UnsignedRawAuthCert: out GenericRawCert);
```

3.8.4.1 Usage

This procedure is called when a user has been granted authorization, and constructs a raw authorization certificate. This will then be ready to be digested and signed by the crypto library.

— `AuthCert` will have been constructed by TIS. The CryptoControl data will have been copied from the user's Privilege certificate, with the DigestID overwritten with 0. This ensures that when the certificate is digested by the crypto library, we will get a new DigestID, and allows the success of signing the Auth Cert to be controlled by SPRE via PrivCert.CryptoControl.DigestInfo.SignReturn.

3.8.4.2 Behaviour

Constructs a raw authorization certificate from its constituent parts.

— `UnsignedRawAuthCert.CertData` will be made up of the unstructured AuthCert, the CryptoControl data, and padding (value *don't care*), in that order.

— `UnsignedRawAuthCert.CertLength` will be set to 212 (the number of bytes in an Authorization Certificate prior to being signed).

3.8.5 AddAuthSignature

```
procedure AddAuthSignature(UnsignedRawAuthCert: in GenericRawCert;
                           SigData: in SignatureData;
                           SignedRawAuthCert : out GenericRawCert);
```

3.8.5.1 Usage

AddAuthSignature is called once a signature has been generated by the crypto library.

— `UnsignedRawAuthCert` is that produced by ConstructAuthCert.

— `SigData` is that produced by the Crypto library Sign operation.

3.8.5.2 Behaviour

It appends the supplied signature it to the certificate data, ready for writing to the user's card.

— `SignedRawAuthCert` is `UnsignedRawAuthCert` with `SigData` appended to the CertData field, and the CertLength field updated to 348.

4 Peripheral Simulator Message Interface

4.1 General Message Structure

The general message structure to be sent by both Praxis and SPRE is as follows:

Direction	ProcID	$ParamID_1$	Value a		Value end		$ParamID_n$	Value a		Value end

i.e. a variable length stream of 32-bit words.

Sections 4.2 to 4.6.1 define the messages to be used for each interface, and includes examples (with variable data in *italics*). Outgoing messages are those from Praxis to SPRE. Incoming messages are those from SPRE to Praxis.

4.1.1 Direction

This is the first 32-bit word of the stream and indicates the direction of the message – incoming or outgoing. Incoming will be represented as 11 00 00 00h; Outgoing will be 00 00 00 00h.

4.1.2 ProcID

ProcID is a unique 32-bit word identifying the procedure call in progress. The MS byte indicates the API:

- 01h for Door;
- 02h for Display;
- 04h for Card Reader;
- 08h for Biometric;
- 10h for Alarm.
- 20h for Latch.

The LS 3 bytes will indicate the procedure/function name. Values will be pre-determined. So, for the Biometric API:

ProcID	Procedure/Function
08 00 00 01h	Initialize
08 00 00 02h	Finalize
08 00 00 04h	SamplePresent
08 00 00 08h	Verify

This should be the same for both incoming and outgoing messages.

4.1.3 ParamID

A 32-bit number representing a parameter of a particular procedure/function (return values of functions are treated as parameters).

So, for Verify in the Biometric API:

ParamID	Parameter
0x001h	Template
0x002h	TemplateLength
0x004h	MaxFAR
0x008h	Matched
0x010h	FARAchieved
0x020h	BioReturn

For outgoing messages, only `in` or `in out` parameters are sent.

For incoming messages, only `in out` or `out` parameters (or return value if a function) are sent.

Ordering of parameters in the message will be numerically increasing.

4.1.4 Value a .. Value end

The data value will be split into 32-bit words for sending. Data will be padded to ensure 32-bit alignment e.g. an unsigned 8 bit value would be sent as [*XX* 00 00 00h].

Boolean values shall be represented as a byte: False – 0; True – any non-zero value.

Characters will be sent as ASCII byte values.

Records, such as GenericRawCert (from the card reader API), will be treated as a single data object. That is, no padding between fields.

Certificates, and data contained within them, will be structured and ordered as defined in the tables in section 2, and will start at offset 0 in the GenericRawCert type.

Large variable length data values, such as a biometric template, will be modeled in the system as a fixed size array (usually of 32-bit words), and have a corresponding variable (*n*) which records the length, in bytes, of the actual data held in the array. For API procedure calls, these will usually be two separate parameters. The first *n* bytes of the array will hold the data, and the rest are *don't care*.

e.g. For the Template parameter of Verify in the Biometric API:

```
Template : TemplateType :=
    (00h, 00h, 01h, E2h, 00h, 00h, 7Fh, FFh, ..)
1st chunk to send will be [00 00 01 E2h]
2nd chunk will be [00 00 7F FFh]
etc.
```

From this, TemplateLength = 482 and MaxFAR = 32767.

As each parameter is a fixed size, the length of a message is pre-determinable, so a message length field is not required.

4.2 Token Reader Interface

4.2.1 ProcIDs

Procedure	ProcID
Initialize	04 00 00 01h
Finalize	04 00 00 02h
ListReaders	04 00 00 04h
GetStatusChange	04 00 00 08h
Connect	04 00 00 10h
Status	04 00 00 20h
Disconnect	04 00 00 40h
GetIDCert	04 00 00 80h
GetPrivCert	04 00 01 00h
GetIACert	04 00 02 00h
GetAuthCert	04 00 04 00h
UpdateAuthCert	04 00 08 00h

4.2.2 Initialize

4.2.2.1 Message Definition

Direction	Message Length (32-bit words)	Parameter(s)	ParamID	Value Size (32-bit words)
Outgoing	2	-	-	-
Incoming	4	ResponseCode	00 00 00 01h	1

4.2.2.2 Example

Outgoing:

```
[00 00 00 00h][04 00 00 01h]
```

Incoming:

```
[11 00 00 00h][04 00 00 01h][00 00 00 01h][00 00 00 00h]
                                          (Success)
```

4.2.3 Finalize

4.2.3.1 Message Definition

Direction	Message Length (32-bit words)	Parameter(s)	ParamID	Value Size (32-bit words)
Outgoing	2	-	-	-
Incoming	4	ResponseCode	00 00 00 01h	1

4.2.3.2 Example

Outgoing:

```
[00 00 00 00h][04 00 00 02h]
```

Incoming:

```
[11 00 00 00h][04 00 00 02h][00 00 00 01h][00 00 00 00h]
                                          (Success)
```

4.2.4 ListReaders

4.2.4.1 Message Definition

Direction	Message Length (32-bit words)	Parameter(s)	ParamID	Value Size (32-bit words)
Outgoing	4	Number	00 00 00 02h	1
Incoming	27	List	00 00 00 01h	20
		Number	00 00 00 02h	1
		ResponseCode	00 00 00 04h	1

4.2.4.2 Example

Outgoing:

```
[00 00 00 00h][04 00 00 04h][00 00 00 02h][00 00 00 02h]
                                          (Expect 2 readers)
```

Incoming:

```
[11 00 00 00h][04 00 00 04h]
[00 00 00 01h][49 4E 54 52h][45 41 44 00h]     (INTREAD)
[45 58 54 52h][45 41 44 00h]                   (EXTREAD)
[..further 16 list chunks..]
[00 00 00 02h][00 00 00 02h]                   (Found 2 readers)
[00 00 00 04h][00 00 00 00h]                   (Success)
```

4.2.5 GetStatusChange

4.2.5.1 Message Definition

Direction	Message Length (32-bit words)	Parameter(s)	ParamID	Value Size (32-bit words)
Outgoing	9	Timeout	00 00 00 01h	1
		Reader	00 00 00 02h	2
		CurrentState	00 00 00 04h	1
Incoming	6	NewState	00 00 00 08h	1
		ResponseCode	00 00 00 10h	1

4.2.5.2 Example

Outgoing:

```
[00 00 00 00h][04 00 00 08h]
[00 00 00 01h][00 00 00 0Eh]                   (Wait 15ms)
[00 00 00 02h][45 58 54 52h][45 41 44 00h]     (EXTREAD)
[00 00 00 04h][00 00 00 04h]                   (Empty)
```

Incoming:

```
[11 00 00 00h][04 00 00 08h]
[00 00 00 08h][FF DD EE 23h]                   (Unrecognized state)
[00 00 00 10h][00 00 00 00h]
```

4.2.6 Connect

4.2.6.1 Message Definition

Direction	Message Length (32-bit words)	Parameter(s)	ParamID	Value Size (32-bit words)
Outgoing	5	Reader	00 00 00 01h	2
Incoming	6	CardHandle	00 00 00 02h	1
		ResponseCode	00 00 00 04h	1

4.2.6.2 Example

Outgoing:

```
[00 00 00 00h][04 00 00 10h][00 00 00 01h]
[49 4E 54 52h][45 41 44 00h]    (Connect to card in 'INTREAD')
```

Incoming:

```
[11 00 00 00h][04 00 00 10h]
[00 00 00 02h][AB CD EF 10h]    (card handle)
[00 00 00 04h][00 00 00 13h]    (card removed)
```

4.2.7 Status

4.2.7.1 Message Definition

Direction	Message Length (32-bit words)	Parameter(s)	ParamID	Value Size (32-bit words)
Outgoing	4	CardHandle	00 00 00 01h	1
Incoming	15	CState	00 00 00 02h	1
		ATR	00 00 00 04h	8
		ResponseCode	00 00 00 08h	1

4.2.7.2 Example

Outgoing:

```
[00 00 00 00h][04 00 00 20h][00 00 00 01h][FE DC BA 98h]
```

Incoming:

```
[11 00 00 00h][04 00 00 20h]
[00 00 00 02h][00 00 00 10h]    (Card is in "specific" state)
[00 00 00 04h][01 23 45 67h]    (Token ID)
[89 AB CD EFh][FF EE DD CCh][BB AA 99 88h][77 66 55 44h]
[33 22 11 00h][01 23 45 67h][89 AB CD EFh]
```

(28 bytes don't care)

[00 00 00 08h][*00 00 00 00h*]

4.2.8 Disconnect

4.2.8.1 Message Definition

Direction	Message Length (32-bit words)	Parameter(s)	ParamID	Value Size (32-bit words)
Outgoing	4	CardHandle	00 00 00 01h	1
Incoming	4	ResponseCode	00 00 00 02h	1

4.2.8.2 Example

Outgoing:

[00 00 00 00h][04 00 00 40h][00 00 00 01h][A*9 4E 54 45h*]

Incoming:

[11 00 00 00h][04 00 00 40h][00 00 00 02h][*00 00 00 01h*]
(invalid handle)

4.2.9 GetIDCert

4.2.9.1 Message Definition

Direction	Message Length (32-bit words)	Parameter(s)	ParamID	Value Size (32-bit words)
Outgoing	4	CardHandle	00 00 00 01h	1
Incoming	258	RawIDCert	00 00 00 02h	251
		StatusOK	00 00 00 04h	1
		ResponseCode	00 00 00 08h	1

4.2.9.2 Example

Outgoing:

[00 00 00 00h][04 00 00 80h]
[00 00 00 01h][*11 22 33 44h*]

Incoming:

```
[11 00 00 00h][04 00 00 80h]
[00 00 00 02h][00 00 00 00h]
[..249 RawIDCert.CertData chunks..]
[00 00 01 E8h]                  (RawIDCert.CertLength = 484 bytes)
[00 00 00 04h][FF 00 00 00h]    (Status is OK)
[00 00 00 08h][00 00 00 00h]
```

4.2.10 GetPrivCert

4.2.10.1 Message Definition

Direction	Message Length (32-bit words)	Parameter(s)	ParamID	Value Size (32-bit words)
Outgoing	4	CardHandle	00 00 00 01h	1
Incoming	258	RawPrivCert	00 00 00 02h	251
		StatusOK	00 00 00 04h	1
		ResponseCode	00 00 00 08h	1

4.2.10.2 Example

Outgoing:

```
[00 00 00 00h][04 00 01 00h]
[00 00 00 01h][11 22 33 44h]
```

Incoming:

```
[11 00 00 00h][04 00 01 00h]
[00 00 00 02h][00 00 00 01h]
[..249 RawPrivCert.CertData chunks..]
[00 00 01 4Fh]                  (RawPrivCert.CertLength < 348 bytes)
[00 00 00 04h][00 00 00 00h]    (Status is not OK)
[00 00 00 08h][00 00 00 00h]
```

4.2.11 GetIACert

4.2.11.1 Message Definition

Direction	Message Length (32-bit words)	Parameter(s)	ParamID	Value Size (32-bit words)
Outgoing	4	CardHandle	00 00 00 01h	1
Incoming	258	RawIACert	00 00 00 02h	251
		StatusOK	00 00 00 04h	1
		ResponseCode	00 00 00 08h	1

4.2.11.2 Example

Outgoing:

```
[00 00 00 00h][04 00 02 00h]
[00 00 00 01h][11 22 33 44h]
```

Incoming:

```
[11 00 00 00h][04 00 02 00h]
[00 00 00 02h][00 00 00 02h]
[..249 RawIACert.CertData chunks..]
[00 00 03 44h]                    (RawIACert.CertLength = 832 bytes)
[00 00 00 04h][FF 00 00 00h]      (Status is OK)
[00 00 00 08h][00 00 00 00h]
```

4.2.12 GetAuthCert

4.2.12.1 Message Definition

Direction	Message Length (32-bit words)	Parameter(s)	ParamID	Value Size (32-bit words)
Outgoing	4	CardHandle	00 00 00 01h	1
Incoming	260	RawAuthCert	00 00 00 02h	251
		Exists	00 00 00 04h	1
		StatusOK	00 00 00 08h	1
		ResponseCode	00 00 00 10h	1

4.2.12.2 Example

Outgoing:

```
[00 00 00 00h][04 00 04 00h]
[00 00 00 01h][11 22 33 44h]
```

Incoming:

```
[11 00 00 00h][04 00 04 00h]
[00 00 00 02h][00 00 00 00h]
[..249 RawAuthCert.CertData chunks..]
[00 00 00 00h]                  (RawAuthCert.CertLength = 0, but...)
[00 00 00 04h][00 00 00 00h]    (Cert does not exist)
[00 00 00 08h][01 00 00 00h]    (Status is OK)
[00 00 00 10h][00 00 00 00h]
```

4.2.13 UpdateAuthCert

4.2.13.1 Message Definition

Direction	Message Length (32-bit words)	Parameter(s)	ParamID	Value Size (32-bit words)
Outgoing	256	CardHandle	00 00 00 01h	1
		RawAuthCert	00 00 00 02h	251
Incoming	6	StatusOK	00 00 00 04h	1
		ResponseCode	00 00 00 08h	1

4.2.13.2 Example

Outgoing:

```
[00 00 00 00h][04 00 08 00h]
[00 00 00 01h][11 22 33 44h]
[00 00 00 02h][00 00 00 03h]
[..249 RawAuthCert.CertData chunks..]
[00 00 01 5Ch]
```

Incoming:

```
[11 00 00 00h][04 00 08 00h]
[00 00 00 04h][00 00 00 00h]    (Status is not OK)
[00 00 00 08h][00 00 00 00h]
```

4.3 Biometric Interface

4.3.1 ProcIDs

Procedure	ProcID
Initialize	08 00 00 01h
Finalize	08 00 00 02h
SamplePresent	08 00 00 04h
Verify	08 00 00 08h

4.3.2 Initialize

4.3.2.1 Message Definition

Direction	Message Length (32-bit words)	Parameter(s)	ParamID	Value Size (32-bit words)
Outgoing	2	-	-	-
Incoming	4	BioReturn	00 00 00 01h	1

4.3.2.2 Example

Outgoing:

```
[00 00 00 00h][08 00 00 01h]
```

Incoming:

```
[11 00 00 00h][08 00 00 01h][00 00 00 01h][00 00 00 01h]
                                          (Internal Error)
```

4.3.3 Finalize

4.3.3.1 Message Definition

Direction	Message Length (32-bit words)	Parameter(s)	ParamID	Value Size (32-bit words)
Outgoing	2	-	-	-
Incoming	4	BioReturn	00 00 00 01h	1

4.3.3.2 Example

Outgoing:

```
[00 00 00 00h][08 00 00 02h]
```

Incoming:

```
[11 00 00 00h][08 00 00 02h][00 00 00 01h][00 00 01 18h]
                                          (ModuleUnload failed)
```

4.3.4 SamplePresent

4.3.4.1 Message Definition

Direction	Message Length (32-bit words)	Parameter(s)	ParamID	Value Size (32-bit words)
Outgoing	2	-	-	-
Incoming	4	*Return Value*	00 00 00 01h	1

4.3.4.2 Example

Outgoing:

```
[00 00 00 00h][08 00 00 04h]
```

Incoming:

```
[11 00 00 00h][08 00 00 04h][00 00 00 01h][0F 00 00 00h]
                                          (Sample is present)
```

4.3.5 Verify

4.3.5.1 Message Definition

Direction	Message Length (32-bit words)	Parameter(s)	ParamID	Value Size (32-bit words)
Outgoing	132	Template	00 00 00 01h	125
		TemplateLength	00 00 00 02h	1
		MaxFAR	00 00 00 04h	1
Incoming	8	Matched	00 00 00 08h	1
		FARAchieved	00 00 00 10h	1
		BioReturn	00 00 00 20h	1

4.3.5.2 Example

Outgoing:

```
[00 00 00 00h][08 00 00 08h]
[00 00 00 01h][00 00 00 C7h]
[..a further 124 template chunks..]
[00 00 00 02h][00 00 00 C7h]    (Template Length is 199 bytes)
[00 00 00 04h][00 00 FF FFh]    (MaxFAR is 65535)
```

Incoming:

```
[11 00 00 00h][08 00 00 08h]
[00 00 00 10h][00 00 00 00h]    (Match not made)
[00 00 00 20h][0E EE EE EEh]    (FARAchieved is 250539758)
[00 00 00 40h][00 00 00 00h]    (No Error)
```

4.4 Door Interface

4.4.1 ProcIDs

Procedure	ProcID
InitializeDoor	01 00 00 01h
FinalizeDoor	01 00 00 02h
GetDoorState	01 00 00 04h

4.4.2 InitializeDoor

4.4.2.1 Message Definition

Direction	Message Length (32-bit words)	Parameter(s)	ParamID	Value Size (32-bit words)
Outgoing	2	-	-	-
Incoming	4	Success	00 00 00 01h	1

4.4.2.2 Example

Outgoing:

```
[00 00 00 00h][01 00 00 01h]
```

Incoming:

```
[11 00 00 00h][01 00 00 01h][00 00 00 01h][FF 00 00 00h]
                                          (Successful)
```

4.4.3 FinalizeDoor

4.4.3.1 Message Definition

Direction	Message Length (32-bit words)	Parameter(s)	ParamID	Value Size (32-bit words)
Outgoing	2	-	-	-
Incoming	4	Success	00 00 00 01h	1

4.4.3.2 Example

Outgoing:

```
[00 00 00 00h][01 00 00 02h]
```

Incoming:

```
[11 00 00 00h][01 00 00 02h][00 00 00 01h][00 00 00 00h]
                                          (Unsuccessful)
```

4.4.4 GetDoorState

4.4.4.1 Message Definition

Direction	Message Length (32-bit words)	Parameter(s)	ParamID	Value Size (32-bit words)
Outgoing	2	-	-	-
Incoming	4	*Return Value*	00 00 00 01h	1

4.4.4.2 Example

Outgoing:

```
[00 00 00 00h][01 00 00 04h]
```

Incoming:

```
[11 00 00 00h][01 00 00 04h][00 00 00 01h][00 00 00 03h]
                                          (Door is open)
```

4.5 Latch Interface

4.5.1 ProcIDs

Procedure	ProcID
InitializeLatch	20 00 00 01h
FinalizeLatch	20 00 00 02h
Unlock	20 00 00 04h
Lock	20 00 00 08h

4.5.2 InitializeLatch

4.5.2.1 Message Definition

Direction	Message Length (32-bit words)	Parameter(s)	ParamID	Value Size (32-bit words)
Outgoing	2	-	-	-
Incoming	4	Success	00 00 00 01h	1

4.5.2.2 Example

Outgoing:

```
[00 00 00 00h][20 00 00 01h]
```

Incoming:

```
[11 00 00 00h][20 00 00 01h][00 00 00 01h][00 00 00 00h]
                                          (Unsuccessful)
```

4.5.3 FinalizeLatch

4.5.3.1 Message Definition

Direction	Message Length (32-bit words)	Parameter(s)	ParamID	Value Size (32-bit words)
Outgoing	2	-	-	-
Incoming	4	Success	00 00 00 01h	1

4.5.3.2 Example

Outgoing:

```
[00 00 00 00h][20 00 00 02h]
```

Incoming:

```
[11 00 00 00h][20 00 00 02h][00 00 00 01h][01 00 00 00h]
                                           (Successful)
```

4.5.4 Unlock

4.5.4.1 Message Definition

Direction	Message Length (32-bit words)	Parameter(s)	ParamID	Value Size (32-bit words)
Outgoing	2	-	-	-
Incoming	0	-	-	-

4.5.4.2 Example

Outgoing:

```
[00 00 00 00h][20 00 00 04h]
```

Incoming:

No Message

4.5.5 Lock

4.5.5.1 Message Definition

Direction	Message Length (32-bit words)	Parameter(s)	ParamID	Value Size (32-bit words)
Outgoing	2	-	-	-
Incoming	0	-	-	-

4.5.5.2 Example

Outgoing:

```
[00 00 00 00h][20 00 00 08h]
```

Incoming:

No Message

4.6 Alarm Interface

4.6.1 ProcIDs

Procedure	ProcID
Initialize	10 00 00 01h
Finalize	10 00 00 02h
Activate	10 00 00 04h
Deactivate	10 00 00 08h

4.6.2 Initialize

4.6.2.1 Message Definition

Direction	Message Length (32-bit words)	Parameter(s)	ParamID	Value Size (32-bit words)
Outgoing	2	-	-	-
Incoming	4	Success	00 00 00 01h	1

4.6.2.2 Example

Outgoing:

```
[00 00 00 00h][10 00 00 01h]
```

Incoming:

```
[11 00 00 00h][10 00 00 01h][00 00 00 01h][FF 00 00 00h]
                                           (Successful)
```

4.6.3 Finalize

4.6.3.1 Message Definition

Direction	Message Length (32-bit words)	Parameter(s)	ParamID	Value Size (32-bit words)
Outgoing	2	-	-	-
Incoming	4	Success	00 00 00 01h	1

4.6.3.2 Example

Outgoing:

```
[00 00 00 00h][10 00 00 02h]
```

Incoming:

```
[11 00 00 00h][10 00 00 02h][00 00 00 01h][00 00 00 00h]
                                                (Unsuccessful)
```

4.6.4 Activate

4.6.4.1 Message Definition

Direction	Message Length (32-bit words)	Parameter(s)	ParamID	Value Size (32-bit words)
Outgoing	2	-	-	-
Incoming	0	-	-	-

4.6.4.2 Example

Outgoing:

```
[00 00 00 00h][10 00 00 04h]
```

Incoming:

```
No Message
```

4.6.5 Deactivate

4.6.5.1 Message Definition

Direction	Message Length (32-bit words)	Parameter(s)	ParamID	Value Size (32-bit words)
Outgoing	2	-	-	-
Incoming	0	-	-	-

4.6.5.2 Example

Outgoing:

```
[00 00 00 00h][10 00 00 08h]
```

Incoming:

No Message

4.7 Display Interface

4.7.1 ProcIDs

Procedure	ProcID
Initialize	02 00 00 01h
Finalize	02 00 00 02h
GetMaxTextSizeTop	02 00 00 04h
GetMaxTextSizeBottom	02 00 00 08h
SetTopText	02 00 00 10h
SetBottomText	02 00 00 20h
SetTopTextScrollable	02 00 00 40h

4.7.2 Initialize

4.7.2.1 Message Definition

Direction	Message Length (32-bit words)	Parameter(s)	ParamID	Value Size (32-bit words)
Outgoing	2	-	-	-
Incoming	4	Success	00 00 00 01h	1

4.7.2.2 Example

Outgoing:

```
[00 00 00 00h][02 00 00 01h]
```

Incoming:

```
[11 00 00 00h][02 00 00 01h][00 00 00 01h][FF 00 00 00h]
                                          (Successful)
```

4.7.3 Finalize

4.7.3.1 Message Definition

Direction	Message Length (32-bit words)	Parameter(s)	ParamID	Value Size (32-bit words)
Outgoing	2	-	-	-
Incoming	4	Success	00 00 00 01h	1

4.7.3.2 Example

Outgoing:

```
[00 00 00 00h][02 00 00 02h]
```

Incoming:

```
[11 00 00 00h][02 00 00 02h][00 00 00 01h][00 00 00 00h]
                                            (Unsuccessful)
```

4.7.4 GetMaxTextSizeTop

4.7.4.1 Message Definition

Direction	Message Length (32-bit words)	Parameter(s)	ParamID	Value Size (32-bit words)
Outgoing	2	-	-	-
Incoming	4	*Return Value*	00 00 00 01h	1

4.7.4.2 Example

Outgoing:

```
[00 00 00 00h][02 00 00 04h]
```

Incoming:

```
[11 00 00 00h][02 00 00 04h][00 00 00 01h][00 00 00 14h]
                                            (20 character Top Line)
```

4.7.5 GetMaxTextSizeBottom

4.7.5.1 Message Definition

Direction	Message Length (32-bit words)	Parameter(s)	ParamID	Value Size (32-bit words)
Outgoing	2	-	-	-
Incoming	4	Success	00 00 00 01h	1

4.7.5.2 Example

Outgoing:

```
[00 00 00 00h][02 00 00 08h]
```

Incoming:

```
[11 00 00 00h][02 00 00 08h][00 00 00 01h][00 00 00 20h]
                                                (32 character Bottom Line)
```

4.7.6 SetTopText

4.7.6.1 Message Definition

Direction	Message Length (32-bit words)	Parameter(s)	ParamID	Value Size (32-bit words)
Outgoing	18	TopText	00 00 00 01h	13
		Length	00 00 00 02h	1
Incoming	4	Written	00 00 00 04h	1

4.7.6.2 Example

Outgoing:

```
[00 00 00 00h][02 00 00 10h][00 00 00 01h]
[49 4E 53 45h][52 54 nn nnh][nn nn nn nnh]     (INSERTdontcare..)
[ ..10 further string chunks ..]
[00 00 00 02h][00 00 00 06h]
```

Incoming:

```
[11 00 00 00h][02 00 00 10h][00 00 00 04h][01 00 00 00h]
```

('INSERT' written)

4.7.7 SetBottomText

4.7.7.1 Message Definition

Direction	Message Length (32-bit words)	Parameter(s)	ParamID	Value Size (32-bit words)
Outgoing	18	BottomText	00 00 00 01h	13
		Length	00 00 00 02h	1
Incoming	4	Written	00 00 00 04h	1

4.7.7.2 Example

Outgoing:

```
[00 00 00 00h][02 00 00 20h][00 00 00 01h]
[53 4D 41 52h][54 20 43 41h][52 44 nn nnh]     (SMART CARDblah..)
[ ..10 further string chunks ..]
[00 00 00 02h][00 00 00 0Ah]
```

Incoming:

```
[11 00 00 00h][02 00 00 20h][00 00 00 04h][00 00 00 00h]
                                          ('SMART CARD' not written)
```

4.7.8 SetTopTextScrollable

4.7.8.1 Message Definition

Direction	Message Length (32-bit words)	Parameter(s)	ParamID	Value Size (32-bit words)
Outgoing	18	ScrollText	00 00 00 01h	13
		Length	00 00 00 02h	1
Incoming	4	Written	00 00 00 04h	1

4.7.8.2 Example

Maximum number of characters on top line is e.g. 20 characters.

Outgoing:

```
[00 00 00 00h][02 00 00 40h][00 00 00 01h]
[46 49 4E 47h][45 52 50 52h][49 45 54 20h]    ('FINGERPRINT '
[54 45 4D 50h][4C 41 54 45h][20 49 53 20h]     'TEMPLATE IS ')
[ ..7 further string chunks ..]               (dontcare..)
[00 00 00 02h][00 00 00 18h]
```

Incoming:

```
[11 00 00 00h][02 00 00 40h][00 00 00 04h][01 00 00 00h]
                                            (text written)
```

A Example Walkthrough

The following two walkthroughs are intended to illustrate how the certificate data is used to drive the internal library stubs. The walkthroughs are based on following information:

— The TIS has been enroled.

— The Crypto library has been initialized.

— TIS is aware of the existence of a CA called “TOK-CA1” (with ID 109) and an AA called “TOK-AA2” (with ID 218).

— The key database contains among others, the following keys:

Key Handle	→ {	Owner;	KeyID;	KeyLength;	IsPublic	}
1	→ {	{ 109; 7; “TOK-CA1” }	20436548	64	True	}
3	→ {	{ 218; 7; “TOK-AA2” }	666	128	True	}
6	→ {	{ 436; 7; “PRAXTIS” }	1	128	False	}

— The privilege certificate RawPrivCert has been read from a user’s token and is in raw form. The data within has the following values:

	RawCert offset (32-bit words)	Field	Subfield	Sub-subfield	Value
CertData	0	CertificateType	-	-	1
	1	Holder	Issuer	ID	109
	2			TextLength	7
	3			Text	“TOK-CA1”
	13		SerialNumber	-	1234
	14	Issuer	ID	-	218
	15		TextLength	-	7
	16		Text	-	“TOK_AA2”
	26	SignatureAlg	-	-	SHA1_RSA
	27	Serial Number	-	-	2468
	28	AttCertValidity	NotBefore	-	1990; 01; 01; 00; 00
	33		NotAfter	-	2004; 01; 01; 00; 00
	38	Privilege	Role	-	0 (User)
	39		Class	-	1 (Unclassified)
	40		Padding	-	*don’t care*
	42	CryptoControl	DigestUpdateReturn	-	‘Ok’
	43		DigestFinalReturn	-	‘Ok’
	44		DigestLength	-	20
	45		Digest	DigestID	4321
	46			SignReturn	‘DeviceError’
	47			VerifyReturn	‘Ok’
	48			Padding	*don’t care*
	53	SignatureData	AlgorithmId	-	SHA1_RSA
	54		SignatureLength	-	128
	55		Signature	KeyID	666
	56			DigestID	4321
	57			Padding	*don’t care*
	86	CertData Padding	-	-	*don’t care*
	250	CertLength	-	-	348

A.1 Verifying a Privilege Certificate

1 CertProcess.ObtainSignatureData is used to extract the signature. This makes use of **CertLength**, and we obtain **SignatureData**.

2 The raw data is then digested:

a CryptoLib.DigestInit is called. A mask is used on **SignatureAlg** to determine the mechanism to use – this returns SHA-1.

b CryptoLib.DigestUpdate is called a total of three times, since the whole certificate (minus signature data) is 212 bytes. The first two times, the ReturnValue is 'Ok'. The final call returns **CryptoControl.DigestUpdateReturn** (which is 'Ok').

c CryptoLib.DigestFinal is called and returns **CryptoControl.Digest** as the produced digest. The DigestLength is set to **CryptoControl.DigestLength**. The DigestLength is valid, all of the certificate was read in, the digest operation was initialized, and the library is initialized, so the ReturnValue is **CryptoControl.DigestFinalReturn**, which is 'Ok'.

3 The digest was 'Ok', so the data is obtained ready to verify the signature:

a The CryptoLib find operations are called to determine the KeyHandle. The template searched for has just one attribute – the owner, which is set to **Issuer**. "TOK-AA2" is known, so find returns the handle for its public key (3).

b RSA is the mechanism to use – determined by applying a mask to **SignatureAlg**.

c The Digest and DigestLength are those produced by step 2, and equal **CryptoControl.Digest** and **CryptoControl.DigestLength** respectively.

d Signature and SigLength are extracted from the SigData obtained in step 1, and equal **SignatureData.Signature** and **SignatureData.SignatureLength** respectively.

4 Call CryptoLib.Verify. ReturnValue is set to **CryptoControl.Digest.VerifyReturn**, since:

a The mechanism (RSA) is recognized and valid.

b **SignatureData.Signature.KeyID** and the KeyID in the CryptoLib match.

c **SignatureData.Signature.DeviceID** and **CryptoControl.Digest.DigestID** match.

d **SignatureData.SignatureLength** is not greater than the KeyLength in the CryptoLib.

e The library is initialized

5 The certificate has been verified. TIS can now use the data contained within the certificate by calling CertProcess.ExtractPrivCertData to extract it. All fields have valid values, so the call is successful.

A.2 Constructing an Authorization Certificate

1 The user has been authorized, so the TIS has produced the data to be put into an authorization certificate. CertProc.ConstructAuthCertificate is called. This copies **PrivCert.CryptoControl**, but overwrites **PrivCert.CryptoControl.Digest.DigestID** with 0.

2 The raw certificate data produced in step 1 is then digested.

a CryptoLib.DigestInit is called. A mask is used on (the TIS generated) AuthCert.SignatureAlg to determine the mechanism to use.

b CryptoLib.DigestUpdate is called a total of three times, since the whole certificate (minus signature data) is 212 bytes. The first two times, the ReturnValue is 'Ok'. The final call returns **CryptoControl.DigestUpdateReturn**.

c CryptoLib.DigestFinal is called. Since the CryptoControl.DigestID is 0, the certificate data is summed modulo 2**32 to produce a new DigestID. The DigestLength is set to **CryptoControl.DigestLength**. The DigestLength is valid, all of the certificate was read in, the digest operation was initialized, and the library is initialized, so the ReturnValue is **CryptoControl.DigestFinalReturn**, which is 'Ok'.

3 The digest was successful, so set up data to attempt a sign:

a A mask is used on (the TIS generated) AuthCert.SignatureAlg to determine the mechanism to use.

b The CryptoLib find procedures are called to obtain a handle for the TIS private key.

4 CryptoLib.Sign is called.

a Signature.KeyID is set to the KeyID of the TIS private key.

b Signature.DigestID is set to the DigestID produced in step 2.

c ReturnValue is set to **PrivCert.CryptoControl.Digest.SignReturn** since the library is initialized.

5 The sign operation has returned 'DeviceError', so the data has not been signed. The user has not been issued an authorization certificate.

Document Control and References

Changes history

Issue 0.1 (14 April 2003): Draft for internal review.

Issue 0.2 (17 April 2003): Provisional for external review.

Issue 1.0 (19 August 2008): Updated for public release.

Changes forecast

Updates following review.

Document references

1 *Interoperability Specification for ICCs and Personal Computer Systems Part 5. ICC Resource Manager Definition*, Bull CP8 et al, Revision 1.0, December 1997.

2 *BioAPI Specification*, The BioAPI consortium, Version 1.1, 16th March 2001.

3 *TIS System Requirements Specification*, S.P1229.41.1

4 *Cryptographic Token Interface Standard*, RSA Laboratories, PKCS #11 v2.11, Revision 1, November 2001