| | |
|---|---|
| Internet Engineering Task Force | D. Lanz |
| Internet-Draft | L. Novikov |
| Intended status: Informational | MITRE |
| Expires: January 26, 2012 | July 25, 2011 |

Common Interface to Cryptographic Modules (CICM)
draft-lanz-cicm-03

## Abstract

This memo presents a programming interface to standardize the way software programs manage cryptographic modules and use cryptographic services offered by modules. Although a number of interfaces for commercial environments have been standardized and are in use, this is the first generic cryptographic interface to be developed that supports cryptographic modules separating two security domains and is thus ideal for the high assurance environment. The interface has been designed to also allow less demanding environments to take advantage of its features.

## Status of this Memo



## Copyright Notice

## Table of Contents

## 1. Introduction

This document defines the high-level entities of a programming interface for high assurance cryptographic modules called Common Interface to Cryptographic Modules (CICM) based on the logical model outlined in [CICM-LM].

### 1.1. Requirements Language

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

### 1.2. Definition Language

CICM is defined using Interface Definition Language (IDL) [IDL], a specification language that describes a software interface in a language-neutral way. IDL compilers can generate a functionally equivalent CICM interface binding for common programming languages. The use of IDL in CICM is not intended to either prescribe or preclude a particular communications protocol such as General Inter-ORB Protocol (GIOP) [CORBA] between programs in different address spaces or on different devices.
Additionally, CICM does not use the IDL exception mechanism to report errors. See Extensions for more information.

### 1.3. IDL Language Mapping Conventions

Memory responsibilities and calling conventions MUST follow the appropriate IDL language mapping conventions.

### 1.4. Endianness

Endianness is the byte ordering used to represent data stored in a computer or transmitted between computers. A big-endian ordering of bytes is REQUIRED by CICM.

### 1.5. Blocking and Non-blocking Calls

All CICM methods block (wait for the operation defined by the method) to complete before returning, unless they are explicitly defined as non-blocking. For example, the CICM::Encrypt::Stream::encrypt method (defined in [CICM-CM]) blocks when sending data on a stream to be encrypted, while its sibling CICM::Encrypt::Stream::encrypt_non_blocking is identified not only in its name as non-blocking, but also clearly within the documentation for the method.

## 1.6. Assumptions

The following assumptions were made in the development of CICM:

- Library implementers may implement part of the specification (refer to the sections on Conformance and Extensions, for the normative rules).

- A client program initiates cryptographic transformations with a cryptographic module via the CICM API. Multiple client programs may concurrently access a single module from a single security domain, but CICM provides no support for controlling access to a module by two or more client programs.

- A module may be implemented as hardware, firmware, or software component, or any combination thereof.

- Although CICM is intended for use in high assurance environments, its use is not precluded in less demanding environments.

- One or more entities between the API library and the module translates CICM commands or interfaces to module-specific commands or interfaces.

- CICM makes no provision in the design of the API to guarantee the confidentiality, integrity, or authenticity of commands and data between a client program calling the API and a module. However, such protections can be applied in the library or runtime system software.

- Specialized hardware (e.g., hardware access tokens, key fill devices, trusted displays) independent of a module may require host (and thus API) interaction or may require no host interaction.

## 1.7. Specification Organization

The CICM specification is composed of five documents.

- [CICM-LM] provides an informative (non-normative) underlying logical model and terminology,

- this document defines the basic types and rules for conformance (Section 6) and extension (Section 7),

- [CICM-MM] defines module management capabilities,

- [CICM-KM] defines key management capabilities, and

- [CICM-CM] defines channel management capabilities.

The informative material is for informational purposes; it assists the reader in the understanding and use of the specification but does not contain provisions required for conformance.
The namespaces, interfaces, datatypes, methods, and attributes that comprise the specification are presented in a prescriptive manner. For each category, each namespace is described followed by the interfaces contained within it. The datatype, method, and attribute definitions then follow each interface definition.

## 2. Fundamental Definitions

### 2.1. Namespace CICM

Namespace CICM

module CICM

CICM is the top-level namespace for all CICM interfaces and sub-namespaces.

### 2.2. Fundamental Types

#### 2.2.1. General Types

Type CICM::UInt32

typedef unsigned long UInt32;

Unsigned 32-bit integer.
Type CICM::Bool

typedef boolean Bool;

Boolean value.
Type CICM::CharString

typedef string CharString;

Sequence of characters.
Type CICM::Buffer

typedef sequence<octet> Buffer;

Byte sequence, encapsulating the sequence of bytes, the length of the sequence, and the amount of allocated space.

#### 2.2.2. Identifiers

Type CICM::ModuleId

typedef CICM::CharString ModuleId;

Unique cryptographic module identifier.
Type CICM::TransId

typedef CICM::UInt32 TransId;

Unique transaction identifier for read/write operations.

### 2.2.3. Status Codes

Type CICM::Status

typedef CICM::UInt32 Status;

Status of an executed method.
See also:

* Appendix Appendix A for a full list of status codes.

### 2.2.4. Classifications

Type CICM::Classification

typedef CICM::UInt32 Classification;

Classification levels.
Constant CICM::C_LEVEL_UNCLASSIFIED

const CICM::Classification
C_LEVEL_UNCLASSIFIED = 0x0000602F;

Value indicating unclassified classification level.
Constant CICM::C_LEVEL_CONFIDENTIAL

const CICM::Classification
C_LEVEL_CONFIDENTIAL = 0x00006029;

Value indicating confidential classification level.
Constant CICM::C_LEVEL_SECRET

const CICM::Classification
C_LEVEL_SECRET = 0x0000602A;

Value indicating secret classification level.
Constant CICM::C_LEVEL_TOP_SECRET

const CICM::Classification
C_LEVEL_TOP_SECRET = 0x0000602C;

Value indicating top secret classification level.

### 2.2.5. Ports

Type CICM::RemotePort

```
typedef CICM::UInt32 RemotePort;
```

Remote module port.
Constant CICM::IMPLICIT_REMOTE_PORT

```
const CICM::RemotePort
IMPLICIT_REMOTE_PORT = 0xFFFFFF99;
```

Value that indicates that the remote port value is implicit.
Type CICM::LocalPort

```
typedef CICM::UInt32 LocalPort;
```

Local module port.
Constant CICM::IMPLICIT_LOCAL_PORT

```
const CICM::LocalPort
IMPLICIT_LOCAL_PORT = 0xFFFFFFBB;
```

Value that indicates that the local port value is implicit.
Constant CICM::FILL_INTERFACE_PORT

```
const CICM::LocalPort
FILL_INTERFACE_PORT = 0xFFFFFFEE;
```

Value that represents the port on which keys are filled or exported.

## 2.3. Fundamental Interfaces

### 2.3.1. Interface CICM::CICMRoot

Interface CICM::CICMRoot

```
interface CICMRoot
```

CICMRoot serves as the entry point to the CICM API and enables a specific cryptographic module of potentially many modules available to a host to be selected.

#### 2.3.1.1. CICM::CICMRoot Methods

Method CICM::CICMRoot::get_module_by_id()

```
CICM::Status get_module_by_id(
in  CICM::ModuleId id,
out CICM::CryptoModule crypto_module_ref
);
```

Returns a reference to the module with the given module unique identifier.
Parameters:

- *[in] id Unique identifier for the module.
- *[out] crypto_module_ref Module associated with the given identifier.

Returns:

- *S_OK, S_GENERAL_ERROR, S_NON_FUNCTIONAL, S_OPERATION_FAILED, S_HOST_RESOURCES, S_INVALID_STATE, S_ALARM_STATE, S_MODULE_NOT_AVAILABLE, S_TIMEOUT, S_MODULE_DOES_NOT_EXIST

Example (C++):

```
CICM::Status sCode;
CICM::CryptoModule device;

// Instantiate the root object.
CICM::CICMRoot cicm = new CICM::CICMRoot();

// Retrieve a reference to the module
// corresponding to the specified module identifier.
const string MODULE_ID = "CM10293495867";

// If found, [device] refers to the specified
crypto module. sCode = cicm.get_module_by_id(MODULE_ID, &device);
```

### 2.3.2. Interface CICM::CryptoModule

Interface CICM::CryptoModule

interface CryptoModule

CICM::CryptoModule contains attributes that provide access to module-specific information and attributes that enable access to module managers, through which nearly all interface functionality is accessed.

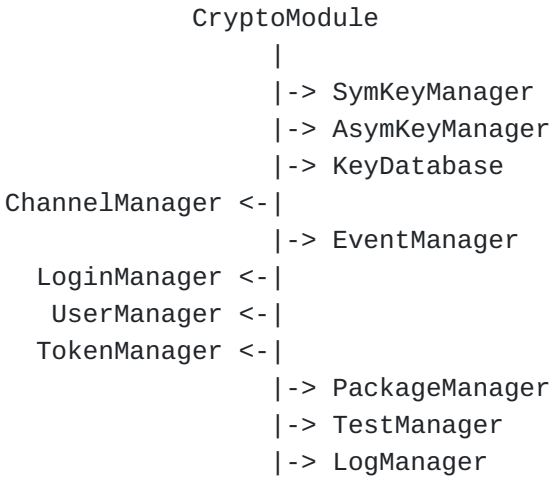


Figure 1. Interface Relationship Diagram for CryptoModule

#### 2.3.2.1. CICM::CryptoModule Attributes

Attribute CICM::CryptoModule::module_id

readonly attribute CICM::ModuleId module_id;

Unique identifier for this module.
Attribute CICM::CryptoModule::manufacturer

readonly attribute CICM::CharString manufacturer;

Name of cryptographic module manufacturer.
Attribute CICM::CryptoModule::model

readonly attribute CICM::CharString model;

Model of cryptographic module.
Attribute CICM::CryptoModule::serial_number

readonly attribute CICM::CharString serial_number;

Serial number of cryptographic module.
Attribute CICM::CryptoModule::module_version

readonly attribute CICM::CharString module_version;

Hardware version of cryptographic module.
Attribute CICM::CryptoModule::software_version

readonly attribute CICM::CharString software_version;

Currently executing software/firmware version number.
Attribute CICM::CryptoModule::driver_version

```
readonly attribute CICM::CharString driver_version;

CICM module-specific abstraction layer version number.
Attribute CICM::CryptoModule::library_version

readonly attribute CICM::CharString library_version;

CICM library version number.
Attribute CICM::CryptoModule::role

readonly attribute CICM::RoleId role;

Current security role in which module is operating.
Attribute CICM::CryptoModule::date_time

attribute CICM::CharString date_time;

Current date/time. Intended for use only with module services that
require coarse-grained time (e.g., timestamp on a log), not for time-
of-day encryption.
Attribute CICM::CryptoModule::sym_key_manager

readonly attribute CICM::SymKeyManager sym_key_manager;

Reference to CICM::SymKeyManager.
Attribute CICM::CryptoModule::asym_key_manager

readonly attribute CICM::AsymKeyManager asym_key_manager;

Reference to CICM::AsymKeyManager.
Attribute CICM::CryptoModule::key_database

readonly attribute CICM::KeyDatabase key_database;

Reference to CICM::KeyDatabase.
Attribute CICM::CryptoModule::channel_manager

readonly attribute CICM::ChannelManager channel_manager;

Reference to CICM::ChannelManager.
Attribute CICM::CryptoModule::event_manager

readonly attribute CICM::ModuleEventManager event_manager;

Reference to CICM::ModuleEventManager.
Attribute CICM::CryptoModule::package_manager

readonly attribute CICM::PackageManager package_manager;

Reference to CICM::PackageManager.
```

Attribute CICM::CryptoModule::token_manager

readonly attribute CICM::TokenManager token_manager;

Reference to CICM::TokenManager.
Attribute CICM::CryptoModule::user_manager

readonly attribute CICM::UserManager user_manager;

Reference to CICM::UserManager.
Attribute CICM::CryptoModule::login_manager

readonly attribute CICM::LoginManager login_manager;

Reference to CICM::LoginManager.
Attribute CICM::CryptoModule::test_manager

readonly attribute CICM::TestManager test_manager;

Reference to CICM::TestManager.
Attribute CICM::CryptoModule::log_manager

readonly attribute CICM::LogManager log_manager;

Reference to CICM::LogManager.

#### 2.3.2.2. CICM::CryptoModule Methods

Method CICM::CryptoModule::configure_fill_interface()

```
CICM::Status configure_fill_interface(
in  CICM::Buffer interface_parameters,
in  CICM::LocalPort fill_port
);
```

Configure a module key fill interface.
Remarks:

- This method accepts an opaque buffer containing a module-specific data structure specifying fill port configuration parameters.

- The format of the interface parameters value is not defined by CICM. The Implementation Conformance Statement (see Section 6) MUST reference a standard format or define a module developer-specific format implemented by the module for this datatype.

Parameters:

- [in] interface_parameters Opaque buffer containing the fill interface configuration parameters.

*[in] fill_port Fill port to configure.

Returns:

*S_OK, S_GENERAL_ERROR, S_NON_FUNCTIONAL, S_OPERATION_FAILED,
S_POLICY_VIOLATION, S_MODULE_RESOURCES, S_HOST_RESOURCES,
S_INVALID_STATE, S_ALARM_STATE, S_MODULE_NOT_AVAILABLE,
S_TIMEOUT, S_NOT_AUTHENTICATED, S_NOT_AUTHORIZED,
S_INVALID_DATA_BUFFER, S_KEY_FILL_DEVICE_NOT_CONNECTED,
S_LOCAL_PORT_INVALID, S_LOCAL_PORT_INCOMPATIBLE,
S_LOCAL_PORT_IN_USE, S_TOKEN_NOT_PRESENT,
S_TOKEN_ADMIN_NOT_PRESENT

Method CICM::CryptoModule::reset()

CICM::Status reset();

Perform a software-initiated reset on the module.
Remarks:

*This method attempts to restart a module in the event of a module
failure or in the event a module has entered an alarm state. A
CICM::S_OK status denotes that the command was accepted by the
module or runtime system, not that any specific action has been
initiated as a result of the reset request.

Returns:

*S_OK, S_GENERAL_ERROR, S_NON_FUNCTIONAL, S_OPERATION_FAILED,
S_POLICY_VIOLATION, S_MODULE_RESOURCES, S_HOST_RESOURCES,
S_INVALID_STATE, S_ALARM_STATE, S_MODULE_NOT_AVAILABLE,
S_TIMEOUT, S_NOT_AUTHENTICATED, S_NOT_AUTHORIZED,
S_TOKEN_NOT_PRESENT, S_TOKEN_ADMIN_NOT_PRESENT

### 2.3.3. Interface CICM::Iterator

Interface CICM::Iterator

interface Iterator

Interface from which other iterators are inherited.
Remarks:

*The specification does not define any specific order in which
iterated elements are returned.

#### 2.3.3.1. CICM::Iterator Types and Constants

Type CICM::Iterator::Status

typedef CICM::UInt32 Status;

Indicates whether or not there are more items over which to iterate.
Constant CICM::Iterator::C_ITERATOR_HAS_NEXT

```
const CICM::Iterator::Status
  C_ITERATOR_HAS_NEXT = 0x00006031;
```

There are more items in the list.
Constant CICM::Iterator::C_ITERATOR_NO_MORE

```
const CICM::Iterator::Status
  C_ITERATOR_NO_MORE = 0x00006032;
```

There are no more items in the list.

#### 2.3.3.2. CICM::Iterator Methods

Method CICM::Iterator::has_next()

```
CICM::Status has_next(
out CICM::Iterator::Status has_next
);
```

Used with get_next() to determine if one or more additional elements are available to be retrieved.
Remarks:

- For elements that have not already been processed, changes in the state of the list/database over which the iterator is being run during the lifetime of the iterator will be reflected in the results from calls to retrieve iterator elements.

Parameters:

- [out] has_next Indicates whether more elements are available to be retrieved.

Returns:

- S_OK, S_GENERAL_ERROR, S_NON_FUNCTIONAL, S_OPERATION_FAILED, S_POLICY_VIOLATION, S_MODULE_RESOURCES, S_HOST_RESOURCES, S_INVALID_STATE, S_ALARM_STATE, S_MODULE_NOT_AVAILABLE, S_TIMEOUT, S_NOT_AUTHENTICATED, S_NOT_AUTHORIZED, S_TOKEN_NOT_PRESENT, S_TOKEN_ADMIN_NOT_PRESENT

## 3. Module Management

Module management capabilities are defined in [CICM-MM].

## 4. Key Management

Key management capabilities are defined in [CICM-KM].

## 5. Channel Management

Channel management capabilities are defined in [CICM-CM].

## 6. Conformance

Many modules will not require the implementation of the full specification to support a module's capabilities. Thus, the CICM conformance model was developed to be flexible. This model does not normatively prescribe the implementation of specific functional subsets of the specification. Instead, CICM outlines a normative Implementation Conformance Statement (ICS) and associated documentation that MUST be supplied with any conformant implementation.

The ICS guides the developer of a library for a specific module to record the implementation state and presence of extensions for each section of the specification. The gradations of the implementation state are relatively coarse: "implemented," "partially implemented," or "not implemented." Extensions are identified as interface extensions or status code extensions, and are recorded as "existing" or "not-existing." An analysis of the resulting matrix enables a software developer using the API or an architect designing a system integrating with a specific cryptographic module to quickly determine if a developer's library will meet user requirements. Those specification sections marked "partially implemented" or for which extensions are indicated may require additional analysis to determine what elements have been extended or are not implemented, and the resulting repercussions on the system utilizing the library.

CICM interfaces are organized into three major sections: module management, channel management, and key management. Each section is partitioned differently into logical subsections in the ICS. The module management section is partitioned into subsections by individual module managers. The channel management section is partitioned into subsections by channel type. The key management section is partitioned into subsections by the type of key and class of operation performed on the key.

An Implementation Data Specification (IDS) based on the ICS also is required. For each implemented interface containing an opaque data parameter (module-specific or infrastructure-specific parameter not described in detail in the specification), the IDS requires a detailed specification of the data structure for each parameter.

An implementation conforms to the specification if it meets the following conditions:

- A CICM library implementation MUST include only the subset of interfaces corresponding to the functionality supported by the

module for which it was designed. The implementation MUST implement the full subset of interfaces implemented by the module. A library MUST implement a non-zero set of interfaces corresponding to functionality implemented by the module that reasonably maps back to the CICM interface and is appropriate for the system in use.

*A CICM library MUST minimally implement the CICMRoot and CryptoModule interfaces, both fundamental parts of the specification without which no other interfaces can be implemented. The CryptoModule interfaces MUST implement minimally one manager, which must be at least "partially" implemented (e.g., simply implementing non-functional inherited or dependent interfaces is non-conformant).

*A CICM library MUST be made available with a corresponding ICS.

*A CICM library MUST be made available with an IDS corresponding to its ICS. The format of any module-specific data structures defined as opaque data elements in the specification with which a client program using CICM must have knowledge MUST be documented by the module developer and MUST be made available as the IDS. If the implementation implements no interfaces with opaque data parameters and includes no extensions, the IDS MUST state that the implementation requires no IDS entries.

*A CICM interface is only conformant if it also implements any inherited and all dependent interfaces (e.g., the interface Encrypt::WithSignConduit defined in [CICM-CM] requires that symmetric keys and asymmetric keysets defined in [CICM-KM] also be implemented). The exception is the CICM::ChannelManager interface, which only requires the implementation of one or more of its inherited interfaces.

*Any interfaces that are not implemented precisely as specified in the normative portion of the specification MUST be identified as extensions to the specification.

*Extensions to the CICM specification MUST NOT contradict nor cause the non-conformance of functionality defined in the normative specification, MUST follow the requirements and guidelines of the normative specification, and MUST be clearly described in supporting documentation.

*Memory responsibilities and calling conventions MUST follow the appropriate IDL language mapping conventions.

### 6.1. Implementation Conformance Statement Contents

A library implementation conforming to the CICM specification MUST be accompanied by an ICS. The ICS is generated by the module developer or implementer of a CICM-conformant library for a specific cryptographic module configuration (including any associated hardware/firmware/software) and MUST contain the following information:

* Details regarding the product and version of the specification to which it conforms, including:
  - CICM version number
  - Product manufacturer/name, version number (hardware, firmware, and software)
  - Configuration details, including patch state
  - Date of claim
* Capability Support Matrix, listing the major sections of the specification and their implementation state ("I"=implemented, "P"=partially implemented, and "N"=not implemented), and the presence of any extensions
* List of developer-defined extensions to specification. Extensions MUST be divided into four classes: Interface extensions, status code extensions, event listener extensions, and constant extensions. Extensions MUST be documented as specified in the IDS.
* List of unique identifiers for all supported cryptographic algorithms, organized by class of algorithm, and all supported key agreement protocols; each algorithm/protocol unique identifier MUST be in CICM-specified format (refer to the section Generating Unique Identifiers).

The following represents a sample CICM ICS.

Date of claim: January 1, 2011 ExampleCorp

# CICMv1 Implementation Conformance Statement

## 1. Product Claiming Conformance

ExampleCorp ABC-XYZ, Version 1.2.3

## 2. Capability Support Matrix

### 2.1 Module Management

Columns
1: Implementation State
2: Interface Extensions
3: Status Code Extensions

| | Impl State | Iface Exts | Codes Exts |
|---|---|---|---|
| EventManager | I | N | N |
| TokenManager | P | N | N |
| LoginManager | I | N | N |
| UserManager | P | I | I |
| TestManager | N | N | N |
| LogManager | N | N | N |
| PackageManager | P | N | N |

### 2.2 Key Management

Columns
1: Implementation State
2: Interface Extensions
3: Status Code Extensions

| | Impl State | Iface Exts | Codes Exts |
|---|---|---|---|
| AsymKeyManager | N | N | N |
| SymKeyManager | P | N | N |
| KeyDatabase | N | N | N |
| KeyProtocol | N | N | N |

### 2.3 Channel Management

Columns
1: Implementation State
2: Interface Extensions | Impl State | Iface Exts | Codes Exts
3: Status Code Extensions

| | | | |
|---|---|---|---|
| EventManager | N | N | N |
| Groups | N | N | N |

### 2.3.1 Encrypt

Columns
1: Implementation State
2: Interface Extensions | Impl State | Iface Exts | Codes Exts
3: Status Code Extensions

| | | | |
|---|---|---|---|
| Encrypt::Stream | N | N | N |
| Encrypt::Controller | N | N | N |
| Encrypt::NegotiatedController | N | N | N |
| Encrypt::Conduit | N | N | N |
| Encrypt::NegotiatedConduit | N | N | N |
| Encrypt::WithMACConduit | N | N | N |
| Encrypt::WithMACNegotiatedConduit | N | N | N |
| Encrypt::WithSignConduit | N | N | N |
| Encrypt::WithSignNegotiatedConduit | N | N | N |
| Encrypt::KeyWrapConduit | N | N | N |

### 2.3.2 Encrypt with Selective Bypass

Columns
1: Implementation State
2: Interface Extensions | Impl State | Iface Exts | Codes Exts
3: Status Code Extensions

| | | | |
|---|---|---|---|
| EncryptBypass::Stream | N | N | N |
| EncryptBypass::Controller | N | N | N |
| EncryptBypass::NegotiatedController | N | N | N |

| | | | |
|---|---|---|---|
| EncryptBypass::Conduit | N | N | N |
| EncryptBypass::NegotiatedConduit | N | N | N |

#### 2.3.3 Decrypt

Columns
1: Implementation State
2: Interface Extensions
3: Status Code Extensions

| | Impl State | Iface Exts | Codes Exts |
|---|---|---|---|
| Decrypt::Stream | N | N | N |
| Decrypt::Controller | N | N | N |
| Decrypt::NegotiatedController | N | N | N |
| Decrypt::Conduit | N | N | N |
| Decrypt::NegotiatedConduit | N | N | N |
| Decrypt::WithMACConduit | N | N | N |
| Decrypt::WithMACNegotiatedConduit | N | N | N |
| Decrypt::WithVerifyConduit | N | N | N |
| Decrypt::WithVerifyNegotiatedConduit | N | N | N |
| Decrypt::KeyUnwrapConduit | N | N | N |

#### 2.3.4 Decrypt with Selective Bypass

Columns
1: Implementation State
2: Interface Extensions
3: Status Code Extensions

| | Impl State | Iface Exts | Codes Exts |
|---|---|---|---|
| DecryptBypass::Stream | N | N | N |
| DecryptBypass::Controller | N | N | N |
| DecryptBypass::NegotiatedController | N | N | N |
| DecryptBypass::Conduit | N | N | N |
| DecryptBypass::NegotiatedConduit | N | N | N |

#### 2.3.5 Duplex

Columns
1: Implementation State
2: Interface Extensions | Impl State | Iface Exts | Codes Exts
3: Status Code Extensions

| | | | |
|---|---|---|---|
| Duplex::Stream | N | N | N |
| Duplex::Controller | N | N | N |
| Duplex::NegotiatedController | N | N | N |
| Duplex::Conduit | N | N | N |
| Duplex::NegotiatedConduit | N | N | N |

#### 2.3.6 Full Bypass (Write)

Columns
1: Implementation State
2: Interface Extensions | Impl State | Iface Exts | Codes Exts
3: Status Code Extensions

| | | | |
|---|---|---|---|
| BypassWrite::Stream | N | N | N |
| BypassWrite::Controller | N | N | N |
| BypassWrite::Conduit | N | N | N |

#### 2.3.7 Full Bypass (Read)

Columns
1: Implementation State
2: Interface Extensions | Impl State | Iface Exts | Codes Exts
3: Status Code Extensions

| | | | |
|---|---|---|---|
| BypassRead::Stream | N | N | N |
| BypassRead::Controller | N | N | N |
| BypassRead::Conduit | N | N | N |

#### 2.3.8 Emit

Columns
1: Implementation State
2: Interface Extensions | Impl State | Iface Exts | Codes Exts
3: Status Code Extensions

| | | | |
|---|---|---|---|
| Emit::RandomController | N | N | N |
| Emit::RandomConduit | N | N | N |
| Emit::PseudorandomController | N | N | N |
| Emit::PseudorandomConduit | N | N | N |
| Emit::KeyStreamGenController | N | N | N |
| Emit::KeyStreamGenConduit | N | N | N |

#### 2.3.9 Integrity

Columns
1: Implementation State
2: Interface Extensions
3: Status Code Extensions

| | Impl State | Iface Exts | Codes Exts |
|---|---|---|---|
| Answer::HashConduit | N | N | N |
| Answer::MACConduit | N | N | N |
| Answer::MACVerifyConduit | N | N | N |
| Answer::SignConduit | N | N | N |
| Answer::VerifyHashConduit | N | N | N |

#### 2.3.10 Single-Domain

Columns
1: Implementation State
2: Interface Extensions
3: Status Code Extensions

| | Impl State | Iface Exts | Codes Exts |
|---|---|---|---|
| Coprocessor::EncryptConduit | N | N | N |
| Coprocessor::EncryptWithMACConduit | N | N | N |
| Coprocessor::EncryptWithSignConduit | N | N | N |
| Coprocessor::DecryptConduit | N | N | N |
| Coprocessor::DecryptWithMACConduit | N | N | N |
| Coprocessor::DecryptWithVerifyConduit | N | N | N |

## 3. Extensions

3.1. Interface Extensions

CICM::UserManager::enable()

CICM::UserManager::disable()

3.2 Status Code Extensions

CICM::S_USER_ALREADY_ENABLED

CICM::S_USER_ALREADY_DISABLED

3.3 Module / Channel Event Listener Extensions

None

3.4 Constant Extensions

None

4. Supported Algorithms

AES128-CBC

3DES-OFB

## 6.2. Implementation Data Specification Contents

The IDS serves as the detailed supporting documentation for the ICS. Conformance with the CICM specification requires that:

- Each implemented interface that accepts an opaque data object MUST reference an existing standard or document the data structure associated with that object in sufficient detail to allow an implementer to create new objects and manipulate existing objects. The exception to this requirement is those cases where a client program will NOT be allowed to manipulate the opaque data object (e.g., CICM::KeyProtocolReceiver::get_from_module defined in [CICM-KM] or CICM::PackageImporter::import_segment defined in [CICM-MM]).

- Each interface extension listed in the ICS MUST be clearly described in the IDS and MUST be documented in a manner similar to the normative CICM documentation.

- Each status code extension listed in the ICS MUST be referenced in the IDS with a corresponding description, numeric code, and a list of CICM interfaces to which the extension applies.

- Each module or channel event listener extension listed in the ICS MUST be referenced in the IDS with corresponding description, numeric code, and data structure definition associated with the event_data parameter, if applicable.

- Each extended constant value listed in the ICS MUST be referenced in the IDS with corresponding description and numeric code.

Examples of interfaces requiring an IDS entry to be conformant include:

- CICM::SymKeyManager::get_key_by_id defined in [CICM-KM], where the key identifier is specific to the key management system in use.

- CICM::LogManager::retrieve defined in [CICM-MM], where the log returned from the method call will vary from module-to-module.

- CICM::ModuleEventListener::event_occurred defined in [CICM-MM], where the event_data parameter passed to a client program as part of an event notification is system specific.

Note that the event listener callbacks (CICM::ModuleEventListener::event_occurred and CICM::ChannelEventListener::event_occurred) require that the event_data parameter be described for each event type implemented.

### 6.3. Generating Unique Identifiers

CICM does not provide a list of algorithms with their corresponding normative unique identifiers. Instead, normative guidance is provided for generating the identifiers for the different classes of algorithms defined in the specification and for key agreement protocols. These identifiers are used by software developers when specifying algorithms or protocols as parameters to CICM methods. This identifier generation guidance is intended to promote interoperability, and encourage the use of the same identifier for algorithms among vendors.
Three major components may be combined to form a unique algorithm identifier: an algorithm (ALGO), that may be precisely specified as an encryption algorithm (ENCRALGO), signature algorithm (SIGALGO), MAC algorithm (MACALGO), or hash algorithm (HASHALGO); a mode (MODE); and an encoding scheme (SCHEME), that may be precisely specified as an encryption scheme (ENCRSCHEME) or a signature scheme (SIGSCHEME). Note that some components above may not apply to certain algorithms. In addition, applicable modes and components need not always be specified. For encryption and signature algorithms, if a length is required, the length SHALL be appended to the algorithm without a dash ("-") delimiter. Otherwise, components are concatenated with a dash ("-"). Alternatively, an identifier can consist of a simple personality designation (PERSONALITY). The personality consists of a combination of parameters that comprise a logically complete crypto, and specifies a

specific equipment type or configuration for which algorithm, mode, and any other parameters are implicit. The designation may contain dashes. Certain algorithms may be appropriate for and thus listed under more than one algorithm class. Below are the classes of algorithms and format of the identifiers for each class:

Asymmetric encryption algorithm identifiers (AsymEncrAlgorithmId)

- *Format: ENCRALGO [ "-" ENCRSCHEME ] | PERSONALITY
- *Examples: "RSA1024-OAEP"

Asymmetric signature algorithm identifiers (AsymSigAlgorithmId)

- *Format: SIGALGO [ "-" HASHALGO [ "-" SIGSCHEME ]] | PERSONALITY
- *Examples: "DSA-SHA1" or "RSA1024-SHA256-PKCS1V1_5"

Symmetric encryption algorithm identifiers (SymEncrAlgorithmId)

- *Format: ENCRALGO | PERSONALITY
- *Examples: "AES128" or "3DES"

Symmetric MAC algorithm identifiers (SymMacAlgorithmId)

- *Format: MACALGO [ - HASHALGO ] | PERSONALITY
- *Examples: "HMAC-SHA1" or "UMAC"

Hash algorithm identifiers (HashAlgorithmId)

- *Format: HASHALGO | PERSONALITY
- *Examples: "MD5" or "SHA1"

Key wrap algorithm identifiers (KeyWrapAlgorithmId)

- *Format: ENCRALGO | PERSONALITY
- *Examples: AESKW

Two major components may be combined to form a key agreement protocol identifier: the key agreement protocol including its version number (KEYAGREEPROTO) and the protocol's associated algorithm suite including its version number (ALGOSUITE). The following is the format for key agreement protocol identifiers.

Key agreement protocol identifier (ProtocolId)

- *Format: KEYAGREEPROTO "-" ALGOSUITE
- *Examples: "IKE2.0-FIREFLY"

Note that the resulting identifiers may not be compatible with those identifiers defined for other module developers' implementations. A client program utilizing an identifier corresponding to one algorithm for a specific module may be required to modify the identifier for the same algorithm for a different type of module. Discrepancies may be discovered through a brief review of the ICS "Supported Algorithms" section.

### 6.4. Conformance Verification

In the future, test assertions may be made available to allow results from different organizations to be compared, and to provide proof of conformance to the specification.

## 7. Extensions

An extension is a mechanism to define functionality beyond what is defined in the official specification. In the interest of promoting interoperability, extensions to the specification are discouraged except where necessary. Extensions to the specification enable module developers to add functionality unanticipated by the specification developers and to support proprietary features.

### 7.1. Extending an Interface

Developers may augment CICM interfaces by extending CICM IDL by adding new methods/attributes to existing interfaces or by deriving off existing CICM interfaces. Extensions SHALL be documented in the ICS.

### 7.2. Extending Codes

CICM codes are constants that share a single 32-bit space. A number of datatypes for different purposes correspond to ranges in this space. The "CICM" codes are normatively defined in the specification; the "extended" codes are module developer-defined extensions. The codes, with their corresponding ranges and uses, are as follows:
CICM status codes

*0x00000000 - 0x00001000

Extended status codes

*0x00001001 - 0x00002000

CICM module event codes

*0x00002001 - 0x00003000

Extended module event codes

*0x00003001 - 0x00004000

CICM channel event codes

*0x00004001 - 0x00005000

Extended channel event codes

*0x00005001 - 0x00006000

CICM generic constants

*0x00006001 - 0x00007000

Extended generic constants

*0x00007001 - 0x00008000

RESERVED

*0x00008001 - 0x7FFFFFFF

Normatively-defined CICM codes SHOULD be used whenever possible. If any of the extended codes above are defined, they MUST be documented as specified below.

### 7.2.1. Extending Status Codes

The return value from CICM methods informs the caller of the status of the call. CICM does not use the IDL exception mechanism to report errors.
The specification normatively defines a set of error codes in the range of 0x00000000 - 0x00001000, which may not be modified or extended. A block of codes in the range of 0x00001001 - 0x00002000 are reserved for module developer-defined status codes. Any codes defined in this range MUST be documented in the ICS.

### 7.2.2. Extending Module/Channel Event Codes

The specification supports registering and unregistering user-defined channel event listeners for specific module and channel events. Module events in the range of 0x00003001 - 0x00004000 and channel events in the range of 0x00004001 - 0x00005000 are normatively defined and may not be modified or extended. A block of module events in the range 0x00003001 - 0x00004000 and channel events in the range of 0x00005001 - 0x00006000 are reserved for module developer-defined events. Any codes defined in this range MUST be documented in the ICS.

### 7.2.3. Extending Constants

A number of constants are normatively defined for specification use in the range of 0x00006001 - 0x00007000. Module developer-defined constants may be specified in the range of 0x00007001 - 0x00008000. Any constants defined in this range MUST be documented in the ICS.

## 8. IANA Considerations

[RFC Editor: Please remove this section prior to publication.]
This document has no IANA actions.

## 9. Security Considerations

This document defines basic aspects of the CICM specification and the normative rules for conformance and extensions. Other aspects of CICM contain important security considerations.

### 9.1. Unauthorized Usage

CICM provides several interfaces related to mitigating unauthorized usage in [CICM-MM]. Furthermore, [CICM-KM] discusses aspects of how authorization can be indirectly controlled via key white lists and black lists.

### 9.2. Inappropriate Usage

CICM defines several status codes related to inappropriate usage. For example, attempting to use an invalid key (S_KEY_INVALID) or specifying an inappropriate algorithm (S_ALGO_INVALID). The wide range of status codes relate to the anticipated mechanisms in which using the interface may fail. Additionally, module developers can extend the set of status codes to accommodate their own needs and prevent inappropriate usage.

## 10. Acknowledgments

Many individuals participated in the development and review of the CICM specification. The CICM development team consists of Ronald Albuquerque, Samuel Cardman, Greg Carrier, James Cottrell, Shirley Kawamoto, Daniel Lanz, Brent Midwood, Lev Novikov, Brian O'Hanlon, Rick Page, Adam Pennington, and Nguyen Thai. The document production team consists of Mark Dwyer, Amanda Lind, and Brian Parrish.
The CICM team wishes to thank the following individuals for participating in a review of the specification:

*Bill Beckwith, Objective Interface Systems

*Dennis Bourget, Viasat

*Thom Brooke, Linquest Corporation

*Randy Culver, RT Logic

*John Davis, ITT Corporation

*Eric Dube, The MITRE Corporation

*Jan Duffy, Rockwell Collins

*H.J. Eckles, General Dynamics

*Mark Flinchbaugh, Harris Corporation

*Carolyn Francisco, The MITRE Corporation

*James Howard, L-3 Communications

*Kent Kofstad, The MITRE Corporation

*Hema Krishnamurthy, ITT Corporation

*Prithvi Kumar, The MITRE Corporation

*Chip McGrogan, L-3 Communications

*Hank Morris, Concurrent Technologies Corporation

*Jeff Picciotto, The MITRE Corporation

*Tom Plachecki, General Dynamics

*Ray Purvis, The MITRE Corporation

*Mike Ridge, The MITRE Corporation

*Harry Shaffer, The MITRE Corporation

*Patrick Smith, Linquest Corporation

*Wayne Staats, Rockwell Collins

*James Steinwachs, Harris Corporation

*Porter Taylor, The MITRE Corporation

*Otaway Thomas, Arkham Technology

*Bob Walcott, The MITRE Corporation

*Blane Yamamoto, SafeNet Mykotronx

## 11. References

### 11.1. Normative References


| | |
|---|---|
| **[RFC2119]** | Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997. |
| **[CICM]** | Lanz, D. and L. Novikov, "Common Interface to Cryptographic Modules (CICM) [RFC Editor: Please update the RFC reference and date prior to publication.]", January 2011. |
| **[CICM-MM]** | Lanz, D. and L. Novikov, "Common Interface to Cryptographic Modules (CICM) Module Management [RFC Editor: Please update the RFC reference and date prior to publication.]", January 2011. |
| **[CICM-KM]** | Lanz, D. and L. Novikov, "Common Interface to Cryptographic Modules (CICM) Key Management [RFC Editor: Please update the RFC reference and date prior to publication.]", January 2011. |
| **[CICM-CM]** | Lanz, D. and L. Novikov, "Common Interface to Cryptographic Modules (CICM) Channel Management [RFC Editor: Please update the RFC reference and date prior to publication.]", January 2011. |
| **[IDL]** | International Standards Organization, "Information technology — Open Distributed Processing — Interface Definition Language", ISO/IEC 14750:1999(E), March 1999. |


### 11.2. Informative References


| | |
|---|---|
| **[RFC3552]** | Rescorla, E. and B. Korver, "Guidelines for Writing RFC Text on Security Considerations", BCP 72, RFC 3552, July 2003. |
| **[CICM-LM]** | Lanz, D. and L. Novikov, "Common Interface to Cryptographic Modules (CICM) Logical Model [RFC Editor: Please update the RFC reference and date prior to publication.]", January 2011. |
| **[CORBA]** | Object Management Group, "Common Object Request Broker Architecture (CORBA) Specification, Version 3.1", January 2008. |


## Appendix A. Status Codes

Each method defined in CICM returns a status value to inform the caller as to the outcome of the call. The documentation for each individual method lists the status codes that may be returned in the event a call to the method results in failure.
The status value CICM::S_OK is returned if a method completes successfully. The output parameters of any methods that return a status other than CICM::S_OK are invalid and MUST NOT be referenced or used.

CICM methods can fail for a variety of reasons, including:

- Invalid, illegal, out-of-range, or poorly formed parameters
- Resources insufficient or unavailable
- Unsupported capabilities
- Policy violation
- Hardware failure.

For additional information concerning extending status codes, see Extensions.
CICM status codes are defined below.

S_OK = 0x00000000

- No error.

S_GENERAL_ERROR = 0x00000003

- Unrecoverable error occurred, potentially leaving module in an inconsistent state.

S_NON_FUNCTIONAL = 0x00000005

- Tamper event or other condition has rendered module non-functional.

S_OPERATION_FAILED = 0x00000006

- Method encountered a general failure, but detailed information about the failure is not available.

S_POLICY_VIOLATION = 0x00000009

- Module policy does not permit the requested action.

S_MODULE_RESOURCES = 0x0000000A

- Module resources necessary to perform the requested operation are not available.

S_HOST_RESOURCES = 0x0000000C

- Host resources necessary to perform the requested operation are not available.

S_INVALID_STATE = 0x0000000F

*Module is in a state that does not allow this operation to be performed.

S_ALARM_STATE = 0x00000011

*Module has entered an alarm state.

S_MODULE_NOT_AVAILABLE = 0x00000012

*Module has been powered down, disconnected, or is otherwise unavailable..

S_TIMEOUT = 0x00000014

*Time to receive response from call exceeded threshold.

S_NOT_AUTHENTICATED = 0x00000017

*User has not authenticated to module.

S_NOT_AUTHORIZED = 0x00000018

*User is not authorized to call method.

S_MODULE_DOES_NOT_EXIST = 0x0000001B

*No module with the specified unique identifier exists.

S_MODULE_IN_USE = 0x0000001D

*Module test initiated when channels or other module resources are in use.

S_NOT_AVAILABLE = 0x0000001E

*Information is not available or cannot be found.

S_INVALID_VECTOR = 0x00000021

*Invalid vector provided; this may be because the length or format of the vector is inappropriate for the algorithm or system with which the vector is being used.

S_INVALID_DATA_BUFFER = 0x00000022

*Data in user-specified buffer parameter is invalid.

S_KEY_USED_INVALID = 0x00000024

*Key specified as parameter to method is invalid; this could denote that the key has been zeroized, a failed parity check, or other conditions that prevent the use of the key.

S_KEY_USED_EXPIRED = 0x00000027

*Key specified as parameter to method has expired and may not be used.

S_KEY_USED_CLASSIFICATION = 0x00000028

*Key specified as parameter to method at wrong classification level.

S_KEY_USED_WRAPPED = 0x0000002B

*Key specified as parameter to method may not be used in the context until it has been unwrapped.

S_KEY_USED_CONTEXT = 0x0000002D

*Attempt to use key in an illegal context as defined by the module; e.g., a key is specified for use on a channel but, due to module architecture, the key is unavailable to that channel.

S_KEY_USED_COMPONENT_NOT_AVAIL = 0x0000002E

*Asymmetric key specified as parameter to method contains only a public key (possibly in a certificate) or only a private key, when the other component is needed by the called method.

S_KEY_INVALID = 0x00000030

*Key is invalid; this could denote that the key has been zeroized, a failed parity check, or other conditions that prevent the use of the key.

S_KEY_EXPIRED = 0x00000033

*Key has expired and may not be used.

S_KEY_INCOMPATIBLE = 0x00000035

*Key type (e.g., TEK, KEK) incompatible with intended usage.

S_KEY_CLASSIFICATION = 0x00000036

*Key at wrong classification level.

```
S_KEY_WRAPPED = 0x00000039

     *Key may not be used in this context until it has been unwrapped.

S_KEY_NOT_WRAPPED = 0x0000003A

     *Key is not wrapped.

S_KEY_NOT_WRAPPABLE = 0x0000003C

     *Module is not able to wrap key.

S_KEY_NOT_EXPORTABLE = 0x0000003F

     *Key is not exportable, potentially because it has not been
      wrapped or other policy disallows it.

S_KEY_WRAPPED_EXISTS = 0x00000041

     *Wrapped key already exists.

S_KEY_UNWRAPPED_EXISTS = 0x00000042

     *Unwrapped key already exists.

S_KEY_UPDATE_MAX = 0x00000044

     *Maximum number of updates for this key has been exceeded.

S_KEY_INVALID_ID = 0x00000047

     *Invalid key identifier specified.

S_KEY_PHYSICAL_LOC = 0x00000048

     *Invalid key physical location specified.

S_KEY_ILLEGAL_CONVERSION = 0x0000004B

     *Target algorithm is incompatible with algorithm associated with
      specified key.

S_KEY_MALFORMED = 0x0000004D

     *Key material supplied is malformed.

S_KEY_METADATA_MALFORMED = 0x0000004E

     *Key metadata supplied is malformed.
```

```
S_KEY_NO_NEXT = 0x00000050

     *No next key available for rollover.

S_KEY_WRONG_TYPE = 0x00000053

     *Illegal attempt to process a symmetric key with an asymmetric
      method or an asymmetric key with a symmetric method.

S_KEY_FILL_DEVICE_NOT_CONNECTED = 0x00000055

     *Key fill device not connected.

S_KEY_FILL_NOT_INITIATED = 0x00000056

     *Manual key fill device interaction not initiated within system-
      defined time limit.

S_KEY_TRUST_ANCHOR = 0x00000059

     *Trust anchor required but is unavailable.

S_LOCAL_PORT_INVALID = 0x0000005A

     *Local port specified is invalid.

S_LOCAL_PORT_INCOMPATIBLE = 0x0000005C

     *Local port specified cannot be used in intended manner.

S_LOCAL_PORT_IN_USE = 0x0000005F

     *Local port specified is currently in use.

S_REMOTE_PORT_INVALID = 0x00000060

     *Remote port specified is invalid.

S_REMOTE_PORT_IN_USE = 0x00000063

     *Remote port specified is currently in use.

S_ALGO_INVALID = 0x00000065

     *Malformed string or unsupported/invalid algorithm specified.

S_ALGO_INCOMPATIBLE = 0x00000066

     *Algorithm incompatible with intended usage (e.g., encryption,
      signature, hashing).
```

S_TOKEN_NOT_PRESENT = 0x00000069

*Token must be inserted to perform the requested operation and no token is available to the module.

S_TOKEN_ADMIN_NOT_PRESENT = 0x0000006A

*Administrator token must be inserted to perform the requested operation and either no token is present or the inserted token is not an administrator token.

S_TOKEN_ACCESS = 0x0000006C

*Token I/O error.

S_TOKEN_RESOURCES = 0x0000006F

*Token resources necessary to perform the requested operation are not available.

S_TOKEN_ASSOC_EXISTS = 0x00000071

*Association between module and token already exists.

S_TOKEN_ASSOC_AT_MODULE = 0x00000072

*Association failed because module will allow no new associations.

S_TOKEN_ASSOC_AT_TOKEN = 0x00000074

*Association failed because token will allow no new associations.

S_TOKEN_ASSOC_NOT_EXIST = 0x00000077

*Association between module and token does not exist at the module, at the token, or both.

S_TOKEN_ASSOC_GENERAL = 0x00000078

*Unspecified token association error occurred.

S_TOKEN_DISASSOC_GENERAL = 0x0000007B

*Unspecified token disassociation error occurred.

S_TOKEN_REC_NOT_FOUND = 0x0000007D

*Specified record not found.

S_TOKEN_TIMEOUT = 0x0000007E

*Timeout for insertion of token has been exceeded.

S_TOKEN_LAST_ASSOCIATED = 0x00000081

*Cannot disassociate the last associated token from this module.

S_PACKAGE_NOT_ACTIVATABLE = 0x00000082

*Specified package is not executable.

S_PACKAGE_ACTIVATED = 0x00000084

*Specified package is currently running.

S_PACKAGE_NOT_ACTIVE = 0x00000087

*Specified package is not currently running.

S_PACKAGE_INVALID = 0x00000088

*Specified package is invalid.

S_PACKAGE_TYPE_INVALID = 0x0000008B

*Specified package type is invalid.

S_PACKAGE_KEY_NOT_AVAILABLE = 0x0000008D

*Package is encrypted and the key specified for use to decrypt package is not available on the module.

S_PACKAGE_KEY_NOT_SPECIFIED = 0x0000008E

*Package is encrypted but no key is specified to decrypt it.

S_LOG_ENTRY_INVALID = 0x00000090

*Log entry is invalid.

S_EVENT_REGISTERED = 0x00000093

*An event has already been registered by this process for this event type.

S_EVENT_NOT_REGISTERED = 0x00000095

*An event has not been registered by this process for this event type.

```
S_EVENT_NOT_SUPPORTED = 0x00000096

     *Event is not supported in this implementation.

S_TRUSTED_DISPLAY = 0x00000099

     *Peer information is available at trusted display.

S_NEGOTIATION_ABORTED = 0x0000009A

     *Negotiation was aborted.

S_NEGOTIATION_FAILURE = 0x0000009C

     *Negotiation failed.

S_NEGOTIATION_IN_PROGRESS = 0x0000009F

     *Negotiation is already in progress.

S_NEGOTIATION_NOT_IN_PROGRESS = 0x000000A0

     *No negotiation has been initiated.

S_NEGOTIATION_TIMEOUT = 0x000000A3

     *Negotiation timed out.

S_CERT_LOCAL_INVALID = 0x000000A5

     *Local certificate used in a key negotiation is invalid; the
      certificate may be corrupted or does not verify.

S_CERT_LOCAL_EXPIRED = 0x000000A6

     *Local certificate used in a key negotiation has expired.

S_CERT_REMOTE_INVALID = 0x000000A9

     *Remote certificate used in a key negotiation is invalid; the
      certificate may be corrupted or does not verify.

S_CERT_REMOTE_EXPIRED = 0x000000AA

     *Remote certificate used in a key negotiation has expired.

S_CERT_REMOTE_PATH = 0x000000AC

     *Certificates to enable verification of remote certificate's
      certification path are not available.
```

S_PROTO_INVALID = 0x000000AF

*Malformed string or unsupported/invalid protocol specified.

S_PROTO_INCOMPATIBLE = 0x000000B1

*Protocol specified is incompatible with intended usage.

S_PROTO_UNDETERMINED = 0x000000B2

*An "implicit" protocol has been specified, but the protocol message does not indicate the protocol.

S_CHANNEL_ERROR = 0x000000B4

*Generic conduit/controller error encountered.

S_CHANNEL_PEER_RESET = 0x000000B7

*Peer crypto reset conduit/controller or conduit/controller ceased operation.

S_CHANNEL_MAX = 0x000000B8

*Limit on total number of conduits/controllers has been reached.

S_CHANNEL_NOT_FOUND = 0x000000BB

*Conduit/controller not found.

S_CHANNEL_IO_ERROR = 0x000000BD

*Conduit/controller I/O error.

S_CHANNEL_DATA_INVALID = 0x000000BE

*Input data to cryptographic operation is invalid (e.g., plaintext for encryption or ciphertext for decryption).

S_CHANNEL_DATA_INVALID_LEN = 0x000000C0

*Plaintext (for encryption) or ciphertext (for decryption) input data to cryptographic operation has an inappropriate length; this could denote that the data is too short, too long, or is not a multiple of some particular block size.

S_CHANNEL_BUFFER_LEN = 0x000000C3

*Output of function is too large for supplied buffer.

S_CHANNEL_IN_GROUP = 0x000000C5

*Conduit/controller already exists as part of group.

S_CHANNEL_CLASSIFICATION = 0x000000C6

*Conduits/controllers are not of the same classification.

S_BYPASS_DATARATE_EXCEEDED = 0x000000C9

*Bypass data rate exceeded.

S_BYPASS_DATALIMIT_EXCEEDED = 0x000000CA

*Bypass data limit exceeded.

S_INTEGRITY = 0x000000CC

*In those cases where an encryption algorithm supplies both confidentiality and integrity (an integrity value is transmitted with the ciphertext), the final decrypt may fail with this integrity error if the integrity check fails.

S_AUTHENTICATION_FAILED = 0x000000CF

*Authentication to the module failed; this could denote that a password is incorrect or that additional authentication data supplied is invalid.

S_USER_AUTHENTICATED = 0x000000D1

*Specified user has already authenticated to module.

S_USERNAME_INVALID = 0x000000D2

*Username is invalid.

S_USER_EXISTS = 0x000000D4

*User already exists.

S_USER_INVALID = 0x000000D7

*User does not exist.

S_ROLE_INVALID = 0x000000D8

*Role does not exist.

S_ROLE_ASSOCIATED = 0x000000DB

*User already associated with this role.

S_ROLE_NOT_ASSOCIATED = 0x000000DD

*User not associated with this role.

S_ROLE_MAX = 0x000000DE

*Maximum number of roles already associated with this user.

S_PASSWORD_INVALID = 0x000000E1

*Specified password does not meet module policy.

S_PASSWORD_INVALID_CHAR = 0x000000E2

*Specified password has invalid characters in it.

S_PASSWORD_INVALID_LEN = 0x000000E4

*Length of specified password is either too long or too short.

S_SALT_INVALID = 0x000000E7

*Invalid salt specified.

S_ITERATION_COUNT_INVALID = 0x000000E8

*Invalid iteration count specified.

S_INSUFFICIENT_ENTROPY = 0x000000EB

*Insufficient entropy available.

## Appendix B. IDL Definitions

```
module CICM {
  typedef unsigned long UInt32;
  typedef string CharString;
  typedef sequence<octet> Buffer;

  typedef CICM::UInt32 LocalPort;
  typedef CICM::UInt32 RemotePort;

  const CICM::LocalPort FILL_INTERFACE_PORT = 0xFFFFFFEE;
  const CICM::LocalPort IMPLICIT_LOCAL_PORT = 0xFFFFFFBB;
  const CICM::RemotePort IMPLICIT_REMOTE_PORT = 0xFFFFFF99;

  typedef CICM::UInt32 Classification;
  const CICM::Classification C_LEVEL_CONFIDENTIAL = 0x00006029;
  const CICM::Classification C_LEVEL_SECRET = 0x0000602A;
  const CICM::Classification C_LEVEL_TOP_SECRET = 0x0000602C;
  const CICM::Classification C_LEVEL_UNCLASSIFIED = 0x0000602F;

  typedef CICM::UInt32 Status;
  const CICM::Status S_OK = 0x00000000;
  const CICM::Status S_GENERAL_ERROR = 0x00000003;
  const CICM::Status S_NON_FUNCTIONAL = 0x00000005;
  const CICM::Status S_OPERATION_FAILED = 0x00000006;
  const CICM::Status S_POLICY_VIOLATION = 0x00000009;
  const CICM::Status S_MODULE_RESOURCES = 0x0000000A;
  const CICM::Status S_HOST_RESOURCES = 0x0000000C;
  const CICM::Status S_INVALID_STATE = 0x0000000F;
  const CICM::Status S_ALARM_STATE = 0x00000011;
  const CICM::Status S_MODULE_NOT_AVAILABLE = 0x00000012;
  const CICM::Status S_TIMEOUT = 0x00000014;
  const CICM::Status S_NOT_AUTHENTICATED = 0x00000017;
  const CICM::Status S_NOT_AUTHORIZED = 0x00000018;
  const CICM::Status S_MODULE_DOES_NOT_EXIST = 0x0000001B;
  const CICM::Status S_MODULE_IN_USE = 0x0000001D;
  const CICM::Status S_NOT_AVAILABLE = 0x0000001E;
  const CICM::Status S_INVALID_VECTOR = 0x00000021;
  const CICM::Status S_INVALID_DATA_BUFFER = 0x00000022;
  const CICM::Status S_KEY_USED_INVALID = 0x00000024;
  const CICM::Status S_KEY_USED_EXPIRED = 0x00000027;
  const CICM::Status S_KEY_USED_CLASSIFICATION = 0x00000028;
  const CICM::Status S_KEY_USED_WRAPPED = 0x0000002B;
  const CICM::Status S_KEY_USED_CONTEXT = 0x0000002D;
  const CICM::Status S_KEY_USED_COMPONENT_NOT_AVAIL = 0x0000002E;
  const CICM::Status S_KEY_INVALID = 0x00000030;
  const CICM::Status S_KEY_EXPIRED = 0x00000033;
  const CICM::Status S_KEY_INCOMPATIBLE = 0x00000035;
  const CICM::Status S_KEY_CLASSIFICATION = 0x00000036;
  const CICM::Status S_KEY_WRAPPED = 0x00000039;
  const CICM::Status S_KEY_NOT_WRAPPED = 0x0000003A;
```

```
const CICM::Status S_KEY_NOT_WRAPPABLE = 0x0000003C;
const CICM::Status S_KEY_NOT_EXPORTABLE = 0x0000003F;
const CICM::Status S_KEY_WRAPPED_EXISTS = 0x00000041;
const CICM::Status S_KEY_UNWRAPPED_EXISTS = 0x00000042;
const CICM::Status S_KEY_UPDATE_MAX = 0x00000044;
const CICM::Status S_KEY_INVALID_ID = 0x00000047;
const CICM::Status S_KEY_PHYSICAL_LOC = 0x00000048;
const CICM::Status S_KEY_ILLEGAL_CONVERSION = 0x0000004B;
const CICM::Status S_KEY_MALFORMED = 0x0000004D;
const CICM::Status S_KEY_METADATA_MALFORMED = 0x0000004E;
const CICM::Status S_KEY_NO_NEXT = 0x00000050;
const CICM::Status S_KEY_WRONG_TYPE = 0x00000053;
const CICM::Status S_KEY_FILL_DEVICE_NOT_CONNECTED = 0x00000055;
const CICM::Status S_KEY_FILL_NOT_INITIATED = 0x00000056;
const CICM::Status S_KEY_TRUST_ANCHOR = 0x00000059;
const CICM::Status S_LOCAL_PORT_INVALID = 0x0000005A;
const CICM::Status S_LOCAL_PORT_INCOMPATIBLE = 0x0000005C;
const CICM::Status S_LOCAL_PORT_IN_USE = 0x0000005F;
const CICM::Status S_REMOTE_PORT_INVALID = 0x00000060;
const CICM::Status S_REMOTE_PORT_IN_USE = 0x00000063;
const CICM::Status S_ALGO_INVALID = 0x00000065;
const CICM::Status S_ALGO_INCOMPATIBLE = 0x00000066;
const CICM::Status S_TOKEN_NOT_PRESENT = 0x00000069;
const CICM::Status S_TOKEN_ADMIN_NOT_PRESENT = 0x0000006A;
const CICM::Status S_TOKEN_ACCESS = 0x0000006C;
const CICM::Status S_TOKEN_RESOURCES = 0x0000006F;
const CICM::Status S_TOKEN_ASSOC_EXISTS = 0x00000071;
const CICM::Status S_TOKEN_ASSOC_AT_MODULE = 0x00000072;
const CICM::Status S_TOKEN_ASSOC_AT_TOKEN = 0x00000074;
const CICM::Status S_TOKEN_ASSOC_NOT_EXIST = 0x00000077;
const CICM::Status S_TOKEN_ASSOC_GENERAL = 0x00000078;
const CICM::Status S_TOKEN_DISASSOC_GENERAL = 0x0000007B;
const CICM::Status S_TOKEN_REC_NOT_FOUND = 0x0000007D;
const CICM::Status S_TOKEN_TIMEOUT = 0x0000007E;
const CICM::Status S_TOKEN_LAST_ASSOCIATED = 0x00000081;
const CICM::Status S_PACKAGE_NOT_ACTIVATABLE = 0x00000082;
const CICM::Status S_PACKAGE_ACTIVATED = 0x00000084;
const CICM::Status S_PACKAGE_NOT_ACTIVE = 0x00000087;
const CICM::Status S_PACKAGE_INVALID = 0x00000088;
const CICM::Status S_PACKAGE_TYPE_INVALID = 0x0000008B;
const CICM::Status S_PACKAGE_KEY_NOT_AVAILABLE = 0x0000008D;
const CICM::Status S_PACKAGE_KEY_NOT_SPECIFIED = 0x0000008E;
const CICM::Status S_LOG_ENTRY_INVALID = 0x00000090;
const CICM::Status S_EVENT_REGISTERED = 0x00000093;
const CICM::Status S_EVENT_NOT_REGISTERED = 0x00000095;
const CICM::Status S_EVENT_NOT_SUPPORTED = 0x00000096;
const CICM::Status S_TRUSTED_DISPLAY = 0x00000099;
const CICM::Status S_NEGOTIATION_ABORTED = 0x0000009A;
const CICM::Status S_NEGOTIATION_FAILURE = 0x0000009C;
```

```
const CICM::Status S_NEGOTIATION_IN_PROGRESS = 0x0000009F;
const CICM::Status S_NEGOTIATION_NOT_IN_PROGRESS = 0x000000A0;
const CICM::Status S_NEGOTIATION_TIMEOUT = 0x000000A3;
const CICM::Status S_CERT_LOCAL_INVALID = 0x000000A5;
const CICM::Status S_CERT_LOCAL_EXPIRED = 0x000000A6;
const CICM::Status S_CERT_REMOTE_INVALID = 0x000000A9;
const CICM::Status S_CERT_REMOTE_EXPIRED = 0x000000AA;
const CICM::Status S_CERT_REMOTE_PATH = 0x000000AC;
const CICM::Status S_PROTO_INVALID = 0x000000AF;
const CICM::Status S_PROTO_INCOMPATIBLE = 0x000000B1;
const CICM::Status S_PROTO_UNDETERMINED = 0x000000B2;
const CICM::Status S_CHANNEL_ERROR = 0x000000B4;
const CICM::Status S_CHANNEL_PEER_RESET = 0x000000B7;
const CICM::Status S_CHANNEL_MAX = 0x000000B8;
const CICM::Status S_CHANNEL_NOT_FOUND = 0x000000BB;
const CICM::Status S_CHANNEL_IO_ERROR = 0x000000BD;
const CICM::Status S_CHANNEL_DATA_INVALID = 0x000000BE;
const CICM::Status S_CHANNEL_DATA_INVALID_LEN = 0x000000C0;
const CICM::Status S_CHANNEL_BUFFER_LEN = 0x000000C3;
const CICM::Status S_CHANNEL_IN_GROUP = 0x000000C5;
const CICM::Status S_CHANNEL_CLASSIFICATION = 0x000000C6;
const CICM::Status S_BYPASS_DATARATE_EXCEEDED = 0x000000C9;
const CICM::Status S_BYPASS_DATALIMIT_EXCEEDED = 0x000000CA;
const CICM::Status S_INTEGRITY = 0x000000CC;
const CICM::Status S_AUTHENTICATION_FAILED = 0x000000CF;
const CICM::Status S_USER_AUTHENTICATED = 0x000000D1;
const CICM::Status S_USERNAME_INVALID = 0x000000D2;
const CICM::Status S_USER_EXISTS = 0x000000D4;
const CICM::Status S_USER_INVALID = 0x000000D7;
const CICM::Status S_ROLE_INVALID = 0x000000D8;
const CICM::Status S_ROLE_ASSOCIATED = 0x000000DB;
const CICM::Status S_ROLE_NOT_ASSOCIATED = 0x000000DD;
const CICM::Status S_ROLE_MAX = 0x000000DE;
const CICM::Status S_PASSWORD_INVALID = 0x000000E1;
const CICM::Status S_PASSWORD_INVALID_CHAR = 0x000000E2;
const CICM::Status S_PASSWORD_INVALID_LEN = 0x000000E4;
const CICM::Status S_SALT_INVALID = 0x000000E7;
const CICM::Status S_ITERATION_COUNT_INVALID = 0x000000E8;
const CICM::Status S_INSUFFICIENT_ENTROPY = 0x000000EB;

interface Iterator {
  typedef CICM::UInt32 Status;
  const CICM::Iterator::Status C_ITERATOR_HAS_NEXT = 0x00006031;
  const CICM::Iterator::Status C_ITERATOR_NO_MORE = 0x00006032;

  CICM::Status has_next(
    out CICM::Iterator::Status has_next );
};
```

```
  typedef CICM::CharString ModuleId;

  interface CryptoModule {
    readonly attribute CICM::ModuleId module_id;
    readonly attribute CICM::CharString manufacturer;
    readonly attribute CICM::CharString model;
    readonly attribute CICM::CharString serial_number;
    readonly attribute CICM::CharString module_version;
    readonly attribute CICM::CharString software_version;
    readonly attribute CICM::CharString driver_version;
    readonly attribute CICM::CharString library_version;
    readonly attribute CICM::RoleId role;
    attribute CICM::CharString date_time;
    readonly attribute CICM::SymKeyManager sym_key_manager;
    readonly attribute CICM::AsymKeyManager asym_key_manager;
    readonly attribute CICM::KeyDatabase key_database;
    readonly attribute CICM::ChannelManager channel_manager;
    readonly attribute CICM::ModuleEventManager event_manager;
    readonly attribute CICM::PackageManager package_manager;
    readonly attribute CICM::TokenManager token_manager;
    readonly attribute CICM::UserManager user_manager;
    readonly attribute CICM::LoginManager login_manager;
    readonly attribute CICM::TestManager test_manager;
    readonly attribute CICM::LogManager log_manager;

    CICM::Status configure_fill_interface(
      in  CICM::Buffer interface_parameters,
      in  CICM::LocalPort fill_port );

    CICM::Status reset();
  };

  interface CICMRoot {
    CICM::Status get_module_by_id(
      in  CICM::ModuleId id,
      out CICM::CryptoModule crypto_module_ref );
  };
};
```

## Authors' Addresses


Daniel J. Lanz Lanz The MITRE Corporation EMail: dlanz@mitre.org

Lev Novikov Novikov The MITRE Corporation EMail: lnovikov@mitre.org