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) Logical Model
draft-lanz-cicm-lm-01

Abstract

This document defines an abstract model for high assurance cryptographic modules. It defines the relevant terminology and high-level descriptions of the types of services and operations offered by such modules. The goal is to provide a common vocabulary for discussing the programming of high assurance modules.
Comments are solicited and should be addressed to the mailing list at cicm@ietf.org.

Status of this Memo

Copyright Notice

Table of Contents

1. Introduction

1.1. Background

Sensitive data is increasingly under attack, whether in transit or at rest. The computer security community has responded to these threats by using cryptography to secure sensitive data. To counter the growing number and types of threats against systems processing sensitive data, module vendors have engineered a diverse set of cryptographic modules. Systems that require cryptographic protection may use various cryptographic services including data encryption, signature generation, hashing, and keystream generation. Cryptographic modules providing these services and the key material they hold must be managed. All of these services have proprietary interfaces that differ significantly among module types, leading to the following problems:

* Replacement of one module type for another and reuse of module-dependent software are inhibited as applications require extensive modifications to adapt to new module types and their proprietary interfaces.

* Developers of systems that host cryptographic modules must accommodate different cryptographic module interfaces for different types of cryptographic modules.

* Test tools and procedures developed for one module usually will not work with other modules.

* Security evaluators must learn multiple module developers' interfaces, increasing evaluation time and expense.

To address these problems, this document outlines a logical model for the Common Interface to Cryptographic Modules (CICM) specification which offers module developers a set of standard programming interfaces for the set of operations supported by high assurance cryptographic modules. Although many Application Programming Interfaces (APIs) intended for commercial cryptography are available, the CICM specification was designed for high assurance environments, but may be used in other environments as well.
Modules do not require changes to support the use of CICM. A module-specific abstraction layer between the library implementing CICM interfaces and the module performs the needed translations between the CICM model of a module and the model presented by a specific module. This abstraction component may be provided by the module developer, a module embedder/integrator, or another interested party. This arrangement is analogous to manufacturers of computer peripheral devices providing platform or operating system-specific drivers for their peripheral devices.

The benefits of using standard interfaces to access cryptographic services include:

*Provide a common, logical model of cryptographic modules that is straightforward to learn and easy to use.

*Enable the replacement of one cryptographic module for another without significant modifications to the client programs that interact and use the cryptographic module, assuming certain compatibility characteristics between the modules exist.

*Accommodate binding to multiple programming languages.

*Enforce the correct use of the API; in particular, interfaces must be used in the intended order, imposable at compile time or recognizable via static analysis.

*Support high assurance evaluation by enabling evaluators to quickly validate that a particular implementation uses only the required functions in the manner they were intended.

1.2. Language Independent

CICM is defined using Interface Definition Language (IDL) [IDL], a specification language that describes a software interface in a language-neutral way. The specification currently does not specify normative bindings for specific programming languages, although bindings for common languages can be generated from the IDL provided with the specification. However, normative bindings for one or more popular programming languages will be made available in a future release of the specification.
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.

1.3. Audience

The CICM specification is written for computer programmers, software engineers, and technical architects with a background in data security and cryptography. Knowledge of object-oriented programming concepts is useful when reading IDL definitions. Software engineers may use the specification when developing software that integrates with cryptographic modules. Technical architects may use the specification when designing systems that incorporate cryptographic modules to secure data within the system or between systems.
Although the specification is targeted to software developers who will access module services using a compliant implementation, it also addresses module developers and others who implement library and other support software.

1.4. Scope of the Specification

The CICM model should provide a common way to access the following services offered by cryptographic modules:

- *Cryptographic module management: Includes retrieving information about a specific module, managing access control, managing module events, and loading and managing software packages on modules.

- *Key management: Includes the generation, storage, protection, and removal of key material, and support for message exchanges used in key agreement and key transfer protocols.

- *Channel management: A channel defines a specific cryptographic transform and encapsulates all attributes associated with that transform. Channel management includes channel instantiation, channel control throughout its lifetime, providing data to a channel for transformation, and extracting transformed data from a channel.

The following elements are not addressed:

- *Hardware interfaces, protocols, or design

- *Details of specific protocols (The model provides a means to move protocol messages into and out of a module, but does not dictate low level protocol.)

- *Internal structure of certain types of data elements (e.g., software packages imported into a module, test results extracted from a module)

- *Policy enforcement (The model provides a means to convey policy elements to the module, but policy enforcement is considered a module responsibility.)

- *Organization of the information stored and processed within a module

- *Separation of commands/data for multiple security domains that access a single CICM library instance (e.g., Multiple Levels of Security).

2. Use Cases

A significant characteristic that differentiates CICM from other cryptographic interfaces such as Cryptoki [PKCS-11], GSS-API [RFC5554], and [JCA] is its ability to support cryptographic modules that separate two security domains. The use cases that follow capture this fundamental element of the CICM model. These use cases can be divided into two basic types:

*Cryptographic transformation of data initiated in one security domain with the result made available in another security domain

*Cryptographic transformation of data within a single security domain: Cryptographic hash or digital signature operations may be initiated in the same security domain where the result is received. Other single domain use cases include data encryption/ decryption for storage and keystream/random data generation.

The data-in-transit and data-at-rest use cases illustrated below incorporate multiple security domains, while the final use case depicts a transformation within a single domain.

2.1. **Data-in-Transit**

The figure below shows a hardware device with an embedded cryptographic module providing encryption and decryption services between a secure and non-secure network. The secure side protocol logic subsystems access cryptographic services using CICM. In this use case, the High Assurance IP Encryptor (HAIPE) device uses CICM to enable the internal protocol logic of the device to access cryptographic services; the network to which the HAIPE device is connected does not interface to the protocol encryptor using CICM.

```
                  High Assurance IP Encryptor
        +---------------------------.-----------------+
        |    HAIPE             Cryptographic  HAIPE   |
Secure <-> Protocol <-> CICM <->  Module <-> Protocol <-> Non-Secure
Network |    Logic                  .         Logic   |   Network
        +---------------------------.-----------------+
                                    .
                             Security Domain
                                Boundary
```

Figure 1. First Data-in-Transit Use Case, HAIPE
The following figure depicts the same use case in its end-to-end configuration.

```
                                      Security Domain
                                         Boundary
                                            .
             +------------------------------.--------+
             |   HAIPE                      .        |
( Secure  ) <-> Protocol <-> CICM <-> Cryptographic <---+
( Network )  |   Logic                    Module     |  |
             +------------------------------.--------+  |
                                                        |
          +-------------> ( Non-Secure ) <--------------+
          |               (  Network   )
          |
          |  +--------.------------------------------+
          |  |        .                     HAIPE    |
          +---> Cryptographic <-> CICM <-> Protocol <-> ( Secure  )
             |      Module                  Logic    |  ( Network )
             +--------.------------------------------+
                      .
                Security Domain
                   Boundary
```

Figure 2. HAIPE Use Case in End-to-End Configuration
A second data-in-transit use case shows a tactical secure radio with an embedded cryptographic module providing encryption and decryption services between a local host and a radio frequency environment. The functional blocks that make up the tactical secure radio are logically identical to those in the first example.

```
                     Tactical Secure Radio
      +----------------------------.---------------------+
Host <-> Information <-> CICM <-> Module <->   Waveform <-> RF
      |  Processing                .          Processing |
      +----------------------------.---------------------+
                                   .
                             Security Domain
                                Boundary
```

Figure 3. Second Data-in-Transit Use Case, Tactical Secure Radio

2.2. **Data-at-Rest**

The figure below shows a cryptographic module providing encryption services for data stored on a disk and decryption services for data read from a disk. A file system driver accesses cryptographic services using CICM. This use case could apply to a laptop computer that contains encrypted data; it would prevent access to sensitive data from a lost or stolen laptop.

```
                    Host                          .
      +----------------------------------+        .
User <->   Word    <->  File  <-> CICM <-> Cryptographic <-> Disk
      |  Processor     System           |       Module
      +----------------------------------+        .
                                                  .
                                            Security Domain
                                               Boundary
```

Figure 4. Data-at-Rest Use Case

2.3. Single Security Domain

The following figure shows a cryptographic transform within a single security domain (it assumes that the transform does not change the classification of the data). The plaintext is conveyed to the module, transformed by an encryption algorithm, and results in ciphertext. This information is then returned to the same domain from which the plaintext originated. Other natural examples of a single domain use case include signing, which results in a digital signature; hashing, which results in a hash value; and keystream generation, which results in keystream data.

```
      1. Query              2. Load
User <---------> Database <---------> Disk
                 Back end
                    |
                    +---------------> Cryptographic
                       3. Decrypt          Module
```

Figure 5. Single Security Domain Use Case

3. Module Management

The fundamental element of the CICM model is the MODULE -- an abstraction which refers to the cryptographic module and its capabilities. A single CICM library may provide access to multiple modules.
Each module contains references to information about the module, including the module manufacturer, serial number, and version numbers. Modules also defines attributes called MANAGERS that provide access to the services made available by the module.

3.1. Managing Module Authentication

Modules may require a host or user to authenticate to the module before the module will enter an operational state, allowing it to accept commands and perform cryptographic transformations. In some cases, a specialized, removable hardware component will perform or participate in the authentication. This hardware component is termed a HARDWARE

ACCESS TOKEN in CICM nomenclature, although other communities may use different terminology. Most implementations that use hardware access tokens will transfer key material between the token and module, independent of CICM. In cases where access tokens are not supported, a user may provide authentication credentials to the module via CICM. In still other cases, support for multi-factor authentication will require a token and a user login. Note that the user and token holder may be different entities.

CICM provides interfaces that can be used separately or in combination with one another as appropriate for the system using them and for the authentication mechanisms offered by the module that is used by the system. Methods to manage module/token associations are available for systems where hardware access tokens are supported. Login methods and related user management methods are supported for systems that require user login.

3.1.1. Managing Hardware Access Tokens

The TOKEN MANAGER defines methods that support associating a token with a module, disassociating a token from a module, and disassociating a module from a token. The manager also supports retrieving a list of token associations on a module and module associations on a token.

3.1.2. Managing Users

The USER MANAGER defines methods that support adding users to and removing users from a module user database, and associating a user with a module-defined ROLE. The manager also supports listing the user database, and the roles defined and supported by the module.

3.1.3. Logging in to a Module from a Host

The LOGIN MANAGER defines methods that enable a user configured on a module to login to and logout from a module.

3.2. Managing Software Packages

The PACKAGE MANAGER defines methods that support importing and managing the executable images that reside on a cryptographic module. These methods enable module software/firmware PACKAGES to be imported and other software package management operations to be performed, including retrieving a list of packages, and activating or deleting a specific package.

The package manager enables packages to be imported into a module in segments rather as an atomic unit. This supports modules that must make special provisions to import executable images due to internal storage space limitations.

3.3. Managing Logs

Modules generate log entries as they operate. The LOG MANAGER defines methods that support retrieving individual log entries or extracting an entire log from a module. Additionally, clients may clear individual log entries or the entire module log.

3.4. Managing Tests

Modules may incorporate built-in tests to validate that module functionality is operating as designed. Some tests may be externally initiated. The TEST MANAGER defines methods that support host-initiated module tests.

3.5. Managing Module Events

The MODULE EVENT MANAGER defines methods that support registering/ unregistering module-generated event notifications received by a client program. Clients can register custom-developed CALLBACK procedures, called LISTENERS, for specific module events. When the condition associated with a specific listener presents itself, the registered listener is called.
Examples of events for which listeners may be registered include:

- Hardware access token has been inserted or removed.
- Module is ready to receive traffic.
- Alarm condition is raised.
- Hardware zeroization condition raised.
- Continuous module/engine health test failed.
- Usable lifespan of key expired.
- Change in module power state.

4. Key Management

Cryptographic modules use key material under their protection as one input to perform a cryptographic transformation. Keys

- can originate at a Key Infrastructure Component that has a trust relationship with the module
- may be agreed upon between the module and another entity
- may be generated on the module itself

     *may be derived from information presented to the module by a
      client program.

Once established on a module, they may be subject to client-initiated management operations or may be used as part of a cryptographic channel to effect cryptographic transformations.
CICM treats SYMMETRIC KEYS and ASYMMETRIC KEYSETS separately. An asymmetric keyset may comprise an asymmetric key pair, the public and private key components of a keypair, the digital certificate corresponding to the keyset public key, one or more verification certificates in the certificate chain of trust, and related public domain parameters.
The ASYMMETRIC KEY MANAGER and SYMMETRIC KEY MANAGER attributes allow for access to asymmetric keysets and symmetric keys, respectively.

4.1. Creating and Establishing Keys

Keys may be moved into a module in one of several scenarios. Each scenario is described in detail below.

4.1.1. No Host Interaction Key Fill

Specialized hardware devices designed to transfer key from a key infrastructure component to a specific cryptographic module may fill key into a module without host involvement and thus no API interaction. In some cases, this process does not support transferring key metadata with a key. This requires host and API interaction to apply metadata to the key inside the module upon completion of the fill.

4.1.2. Client Program-Initiated

In some cases, key fill devices require host interaction to initiate a key fill. In such cases, the target key storage location or key tagging information can be specified prior to the initiation of the fill.
Keys may be imported directly or derived using a text-based secret provided by the user of the client program. Keys also may be generated on the module. Each case results in a persistent key.
A key also is implicitly established each time a channel is created using an asymmetric keyset and upon renegotiation. Keys resulting from channel-based key agreement are ephemeral; they are not generally managed outside of a channel. Ephemeral keys also may be destroyed when a channel is destroyed.

4.1.3. Module/Key Infrastructure Initiated

A facility to operate a key agreement protocol with an infrastructure component is supported. This facility also enables key material or key revocation information to be authenticated by one of the module's trust anchors, and then loaded into the module.

4.2. Exporting Keys

Methods to export key material out of a module are supported. A module may require wrapping the key material prior to export or may disallow this operation.

4.2.1. Locating and Retrieving Information about a Key

A method to locate a specific key on a module based upon identification information associated with the key is supported. In addition, the entire key database may be listed.

4.2.2. Applying Metadata to Keys

Key metadata may be retrieved and set for individual keys. Metadata elements include the key identifier, alias, and classification. Untagged keys that are imported via a fill device may require certain metadata to be applied after the conclusion of the load.

4.2.3. Performing Operations on Keys

A number of management operations on keys are supported. Keys may be wrapped (cryptographically protected) in preparation for export, or unwrapped after import. All key material on the module can be zeroized either on a key-by-key basis or as a whole. There are also operations to perform key conversions and updates.

4.2.4. Enabling Remote Management

CICM supports various key management-related protocol messages including remote key functions (e.g., remote zeroize or rekey), infrastructure-initiated key revocation, and trust anchor management.

5. Channel Management

The CHANNEL is the fundamental construct under which one or more related cryptographic transforms are performed, and within which all details and attributes associated with the transform are encapsulated, including the path through the module. Most channels accept data from a port in the local security domain, transform the data, and output the result on a port in another security domain. A channel also may perform transformations within a single security domain, or may accept data for transformation in one domain and output the result in another. The channel type determines which ports must be specified when a channel is created.

```
               Security Domain         Security Domain
                  Boundary                Boundary
                     .                       .
            +--------.--------+     +--------.--------+
 Client  --(a) Cryptographic (b)---(c) Cryptographic (d) -- Client
Program X   |      Module     |     |      Module     |    Program Y
            +--------.--------+     +--------.--------+
                     .                       .

            X local    X remote     Y remote    Y local
            port (a)   port (b)     port (c)   port (d)
```

Figure 6. Local and Remote Port Nomenclature for Channels that Operate in Two Security Domains

Three classes of objects are fundamental to the creation and use of CICM channels. A CONTROLLER is used to configure and control a channel. A STREAM enables data to be sent to a module to be transformed, and transformed data to be received using a controller as a foundation. A CONDUIT is the sum of a controller and a stream. Thus, the term "channel" is only an abstraction representing the logical path through the module on which cryptographic transformations are performed.

```
+----------------------------------------+
|                  Channel               |
|                                        |
|  +-----------------------+             |
|  |             +--------------------+  |
|  | Controller  | Conduit |   Stream |  |
|  +-----------------------+          |  |
|                +--------------------+  |
+----------------------------------------+
```

Figure 7. Relationship Between Channel, Conduit, Controller, and Stream

This division of responsibility makes channels very flexible. One client program can be responsible for creating and managing channels with a controller, and another can send data over this pre-configured channel for transformation using a stream. In some environments, data to be transformed never enters the host to pass through the API. Instead, it is clocked directly through the module. In this situation, a controller is configured, but no stream is configured since it would never be used. In other cases, a client program is required to configure the channel and pass data through the channel it configured. In this case, the client program configures a conduit, which incorporates a controller and a stream.

Both controllers and conduits accept symmetric keys, requiring that the client program configuring the channel and its remote peer share the same secret key. Alternatively, all peers may hold their own respective asymmetric keysets, requiring a key negotiation which, upon successful completion, results in each peer holding an ephemeral symmetric key.

CICM supports a NEGOTIATOR for this purpose. A successful negotiation results in a negotiated controller or conduit.
CICM supports the following channel types:

- Encryption/decryption, including selective bypass
- Signature generation/verification
- Message Authentication Code (MAC) generation/verification
- Cryptographic hashing
- Keystream generation
- Random/pseudo-random data generation
- Key wrap
- Full bypass.

CICM also supports hybrid channel types. A channel that simultaneously supports encryption and signature, resulting in both ciphertext and a final signature value, is a hybrid channel.
Each of the types above differs in the way it is configured, its configuration options, and how it handles the cryptographic transformation of data. Consider the following examples portraying the diversity of the channel types:

- The encryption channel accepts plaintext to be transformed, and can return the resulting ciphertext directly to the caller or route it a different security domain
- The random data generation channel requires no data for transformation, but emits a random stream
- The signature channel accepts an indeterminate amount of data, and returns an algorithm-specific fixed-sized value
- The hashing channel does not accept a cryptographic key as a parameter, as most of the other channel types do (keyed hashes are supported by MAC channels)
- The decryption channel accepts a state vector input parameter, but does not allow a state vector to be generated.

This diversity results from the fundamental characteristics of the cryptographic primitives that are being abstracted. The CHANNEL MANAGER defines the methods that support creating conduits, controllers, streams, and negotiators for each of the channel services listed above.

5.1. Creating Channels

Creating a channel requires an awareness of the options available:

- *The type of cryptographic operation desired (encryption, hashing, keystream generation, etc.)

- *How the channel will be used (control-only, send/receive data only, or both control and send/receive data)

- *The type of key that will be used for channels that require a symmetric key or an asymmetric keyset (hybrid channels accept two keys).

Selecting among these options enables the client program developer to determine what channel interface to use.
In cases where an asymmetric keyset is used, channels are created as a result of a key agreement protocol negotiation with a remote peer. To ensure that it is the expected peer, a human user at the client may validate information extracted from the peer's certificate. If the module uses a trusted display, the module communicates the peer information directly to the display. Based upon user input at the display, host-independent negotiation is continued or aborted. If no trusted display is available, the client program requests information about the remote peer, displays it at the host for user confirmation, and provides positive confirmation via the API that the peer is valid, allowing the negotiation to continue.

5.1.1. Encryption and Decryption

CICM defines interfaces to support encryption and decryption between two security domains or within a single security domain. Additional variants are defined including hybrid channels that can concurrently compute integrity values. Another set of variants provides methods to perform encryption/decryption with selective bypass.
If an asymmetric keyset is used to create a channel, a negotiation process is initiated, which results in a negotiated channel. Negotiated versions of hybrid channels also are available. For those negotiator versions that combine encryption with integrity value generation, negotiation applies only to the encryption key specified when the channel is negotiated, not the signature or MAC key.
Channel-based multiple key wrap/unwrap support is provided via a special channels for that purpose.
CICM also supports encryption/decryption channels that operate in coprocessor mode . These channels accept their input and return their output as part of the same method call. Where relevant, the integrity value or verification status (verified/not verified) is returned when the final block of the input has been presented for transformation.

Duplex channel configurations that use the same key to perform encrypt and decrypt transformations also are supported. Negotiated versions of the duplex channel also are available.

5.1.2. Bypass

Bypass channels capable of defining a path through a module and then bypassing data from one security domain to a different domain are supported. Selective bypass also is supported on encryption and decryption channels.

5.1.3. Integrity

Interfaces to compute and validate integrity values using asymmetric key-derived digital signatures or symmetric key-derived MACs are available. A variant on the sign and verify interfaces accepts a previously generated hash value in place of a message.

5.1.4. Hashing

A channel to calculate a fixed-length cryptographic hash from an input message is available. Keyed hashes are supported by MAC channels.

5.1.5. Keystream Generation

Channels are supported to read keystream from a module.

5.1.6. Random Data

Separate interfaces are defined to retrieve random or pseudorandom data from a module.

5.2. Managing Channels

Only conduits and controllers (not streams) can manage channels.
Negotiators can manage the negotiation aspects of a channel.
The management operations that can be performed on a channel are specific to each channel type, but the following general operations are supported:

*Generating, extracting, and setting state vectors

*Resynchronization

*Initiating a key rollover

*Initiating a key update.

Negotiators support the following general operations:

- *Renegotiation
- *Changing classification level/acknowledging change of classification level.

Managing state vectors is an important channel management capability. CICM provides a method to explicitly generate a state vector for those algorithms/modes that require a random initialization vector (IV), although modules may alternatively generate an IV as a byproduct of channel creation. CICM also provides a method to set the state vector on a channel. This may be used to:

- *Set the decrypt channel to the IV generated/used on the encrypt side of a channel.
- *Provide a vector on a block-by-block basis for appropriate algorithms/modes or at each time epoch (e.g., time-of-day encryption). In addition, a method is available to take a special state vector called a synchronization vector to assist in resynchronizing a channel.

5.3. Using Channels

Only conduits and streams (not controllers) can send data for transformation and receive cryptographically transformed data on a channel.
The data operations that can be performed on a channel or stream are specific to each channel type, but the following general operations are supported:

- *Sending data on a channel to initiate a cryptographic transformation:
  - -Blocking send: Call does not return until data has been sent or the operation times out.
  - -Non-blocking send: Call queues data for sending and returns immediately to the caller.
  - -Poll: Determines status of non-blocking send operation.
- *Receiving transformed data from a channel:
  - -Blocking read: Blocks until data becomes available or the operation times out.
  - -Non-blocking read: Call queues a buffer to receive data and returns immediately.

-Poll: Determines status of non-blocking read operation.

-Notification via callback that data has become available using a channel event listener.

Although it is possible for multiple client programs to use the same stream, the model provides no facilities to coordinate the parties participating in the communication.
Certain channel services support receiving an "answer" from a channel. For example, signature and hashing channels accept variable amounts of data for transformation before returning a final, constant-sized "answer" (a signature or a hash) to the caller. HYBRID CHANNELS require sending/receiving data and receiving a final "answer" after a discrete unit of data has been transformed.
The figure below depicts the use of a hybrid channel. Plaintext is sent through CICM for transformation. The module performs encrypt and sign transformations on the plaintext data. Ciphertext resulting from the encrypt transform emits from the module in a different security domain than the one in which it originated. When it is finished presenting data for transformation, the client program requests the signature that results from the transaction.

```
                         .
        Plaintext        .       Ciphertext
  Host  --------->  Cryptographic ---------->
        <---------      Module
         Signature       .
                         .
                    Security Domain
                       Boundary
```

Figure 8. Hybrid Sign-Encrypt Channel Operations
Each type of channel supports a specific set of channel data operations. Channel types and the data operations they support are listed below:

*Encrypt, selective bypass with encryption, and full bypass write channels: Write data in the local security domain for transformation and output in another security domain.

*Decrypt, selective bypass with decryption, and full bypass read channels: Read transformed data from one security domain into the local security domain.

*Coprocessor channels: Data is presented for transformation and the result received within the same security domain.

*Duplex channels: Read/write exchange between two security domains.

* Keystream and random data generation: Transformation within module results in data stream that emits in the local domain.

5.4. Grouping Channels

Controllers and conduits can be grouped to enable certain characteristics to be shared. One characteristic may be the state vector associated with the channels. This supports environments where two or more channels with related security rules supporting a single operation are used within a system. Whenever a shared characteristic is changed on a controller or conduit in a group, the effect of this change is applied to all controllers/conduits in the group.

5.5. Receiving Notification of Channel Events

The CICM model defines methods that support managing module event notifications. Similar support is available at the granularity of an individual conduit/controller. Conduits and controllers define methods that support registering/unregistering channel-specific module-generated event notifications captured by a client program. Clients can register custom-developed callback procedures called listeners for specific channel events. When the condition associated with a specific listener presents itself, the registered listener is called. Examples of channel events for which listeners may be registered include:

* Data is available.

* Synchronization with peer has been lost.

* Remote peer no longer available.

* General channel error encountered.

5.6. Destroying Channels

Conduits and controllers may be destroyed when their services are no longer needed. A channel is destroyed without regard for users who may have pending operations on the channel. Any ephemeral keys associated with the channel also may be destroyed. A stream ceases to function when its associated controller is destroyed. A destroyed channel is removed from any channel groups to which it belongs without effect upon other controllers/conduits in the group.

6. IANA Considerations

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

7. Security Considerations

7.1. Unauthorized Usage

Cryptographic modules are typically protected assets and most have built-in mechanisms for preventing unauthorized usage. Section 3.1 discusses several mechanisms for preventing unauthorized usage including using hardware access tokens and user login. Specific implementations may also consider the use of access control lists.

7.2. Inappropriate Usage

The log manager described in Section 3.3 may be used by some modules to report certain types of usage which may act as a type of audit log thereby providing information about inappropriate usage.

7.3. Confidentiality and Data Integrity

Many of CICM's channel types provide confidentiality services such as encryption and decryption as well as data integrity services such as hashing, MACing, signing, and verifying.
Hybrid channels provide a combination of confidentiality and data integrity service.

7.4. Bypass

There are four broad categories of bypass that are supported: reading and writing full bypass; encrypting and decrypting with selective bypass. These services must be used with caution in order to avoid accidental or malicious bypass of protected data.

7.5. Entity Authentication

As described in Section 5, a negotiator is the primary mechanism for establishing that the peer entity is the one desired.
Similarly, the mechanism described in Section 4.1.3 ensures that the infrastructure component that is attempting to send data to the module is trusted.

8. References

[RFC4949]	Shirey, R., "Internet Security Glossary, Version 2", RFC 4949, August 2007.
[RFC5554]	Williams, N., "Clarifications and Extensions to the Generic Security Service Application Program Interface (GSS-API) for the Use of Channel Bindings", RFC 5554, May 2009.
[IDL]	International Standards Organization, "Information technology — Open Distributed Processing — Interface

	Definition Language", ISO/IEC 14750:1999(E), March 1999.
[CORBA]	Object Management Group, "Common Object Request Broker Architecture (CORBA) Specification, Version 3.1", January 2008.
[CNSSI-4009]	Committee on National Security Systems (U.S. Government), "National Information Assurance (IA) Glossary", CNSS Instruction No. 4009, revised June 2006.
[FIPS-140-2]	Federal Information Processing Standards Publication (FIPS PUB) 140-2, "Security Requirements for Cryptographic Modules", May 2001.
[PKCS-11]	RSA Laboratories, "PKCS #11 v2.30: Cryptographic Token Interface Standard", April 2009.
[GCS-API]	The OpenGroup, "Generic Cryptographic Service API", June 1996.
[JCA]	Oracle, "Java Cryptography Architecture", August 2002.

Appendix A. Terms

This section contains a list of terms and their corresponding definitions as used in this model. Following the formatting convention in [RFC4949], each term is preceded by a dollar sign ($) and a space to facilitate automated searching.
$ alarm

- *Output signal that denotes that the module has entered an alarm state. An alarm condition may prohibit a module from performing cryptographic operations.

$ asymmetric key

- *Pair of related keys, a public key known to everyone and a private key known only to the owning entity. See symmetric key and asymmetric keyset.

$ asymmetric keyset

- *May comprise one more of the following components: an asymmetric key pair, the public and private key components of a keypair, the digital certificate corresponding to the keyset public key, one or more verification certificates in the certificate chain of trust, and related public domain parameters. See also asymmetric key.

$ asynchronous notification

*Delivery of an indication of a condition or event where, from the point of view of the recipient (the client program), the delivery occurs asynchronously via a callback. See also event and event notification.

$ attribute

*State associated with an instance of an interface.

$ authentication

*Security measure designed to establish the validity of a transmission, message, or originator, or a means of verifying an individual's authorization to receive specific categories of information ([CNSSI-4009]).

$ authorization

*Access privileges granted to a user, program, or process ([CNSSI-4009]).

$ blocking

*A call to a method is blocking if the method does not return program control to the caller until either the operation has completed or an error is recognized. See also non-blocking.

$ buffer

*Collection of binary data.

$ bypass

*In cryptography, this is an operation whereby all of the data is passed from one security domain through the cryptographic module to the other security domain without having a cryptographic transformation applied to it. See also selective bypass.

$ callback

*Procedure provided by the client program that is to be invoked when an appropriate condition or event is recognized. See also asynchronous notification.

$ certificate

*Digitally signed document that binds a public key with an identity. The certificate contains, at a minimum, the identity of the issuing Certification Authority, the user identification information, and the user's public key ([CNSSI-4009]).

$ certificate revocation list (CRL)

*List of certificate serial numbers corresponding to certificates that have been revoked or are no longer valid.

$ channel

*Abstraction under which one or more cryptographic transforms are performed and within which all details associated with the transform are encapsulated, including the path through the module. See also conduit, controller, and stream.

$ channel type

*Cryptographic transform to be applied on a channel.

$ client program

*Program linked to a CICM library running as a single process on a host computer system that accesses cryptographic services and/or to manages a cryptographic module.

$ conduit

*Abstraction that encapsulates channel control and data flow. See also channel, controller, and stream.

$ controller

*Abstraction used to configure and control a channel. See also channel, conduit, and stream.

$ coprocessor mode

*Mode of operation in which cryptographic transformations are performed within a single security domain. For example, in coprocessor mode, a client program provides plaintext to a module, the plaintext is transformed, and the resulting ciphertext is returned to the same client program.

$ cryptographic module

*Abstraction of hardware, firmware, or software components that makes cryptographic services available to client programs via one or more channels.

$ cryptographic synchronization

*Process by which a receiving decrypting cryptographic logic attains the same internal state as the transmitting encrypting logic.

$ cryptographic transform

*The specific cryptographic process that is to be applied to a stream of data or is used to generate keystream or random data, often (but not always) based upon a cryptographic key. Transforms include encryption, decryption, signing, keystream generation, hashing, and random data generation.

$ driver

*Conceptual component residing on a host that enables the exchange of commands and data between the client program and a module. Module-specific abstraction layer that serves as a translation mechanism between the individual functions defined in the CICM library and the commands specific to a given cryptographic module. This component also provides a conduit for data between a host and a module.

$ ephemeral symmetric key

*Symmetric cryptographic key generated as part of a key negotiation process. Ephemeral keys may be destroyed when the channel or session utilizing the ephemeral key completes. Ephemeral keys are not visible if a client program lists the keys on a module. See also static key.

$ event

*Situation occurring on a module or a channel for which a client program may be notified.

$ event notification

*Call from the host runtime system to a client program announcing that a specific situation has occurred. See callback and asynchronous notification.

$ grade

*Negotiated classification level of a channel.

$ hardware access token

*Removable device used to provide locking and unlocking features for the cryptographic capabilities of a cryptographic module.

$ host

*Computer system upon which a client program linked to a CICM library executes.

$ hybrid channel

*Channel that simultaneously supports two fundamental cryptographic services; for example, an encryption with signature channel transforms data, resulting in both ciphertext and a final signature value.

$ iterator

*Software construct that enables a software program to walk through a list of related items.

$ key

*See symmetric key and asymmetric key.

$ key agreement protocol

*Protocol that allows two or more participants to negotiate an ephemeral symmetric key without disclosing the resulting key material to non-participants. The protocol is conducted in such a way that all participants influence the outcome.

$ key encryption key

*Key that encrypts or decrypts another key for transmission or storage.

$ key fill device

*Devices that read-in, transfer, and store key material.

$ key fill interface

*Set of protocols, electrical connections, and physical characteristics that comprise the connecting link between a key fill device and a cryptographic module. CICM enables a key fill interface to be configured and actions to be initiated on a key fill device via the key fill interface.

$ key infrastructure

*Set of hardware, software, people, policies, and procedures needed to create, manage, store, distribute, and revoke key material.

$ key rollover

*Process of moving from one key to another in a pre-defined sequence of keys; may also be referred to as "key supersession."

$ key tag

*Identification information associated with certain types of electronic key ([CNSSI-4009]).

$ key unwrap

*Process whereby an encrypted cryptographic key is decrypted using a cryptographic module and a different key.

$ key update

*Deterministic one-way transformation of a symmetric key (and its current update count) to a new key.

$ key wrap

*Process whereby a cryptographic key is encrypted by a cryptographic module using a separate key in a manner sufficient to protect the key at the level of its classification.

$ keystream

*Sequence of symbols produced by a cryptographic module using a cryptographic key to combine with plain text to produce cipher text, control transmission security processes, or produce key ([CNSSI-4009]).

$ listener

*Method registered by the client program that will be called upon the occurrence of a specific module event.

$ local port

*Port on module in same security domain in which client program is located to which commands are presented and through which data is sent/received. See also remote port.

$ manager

*Specialized attributes that encapsulate different classes of module, key and channel management functionality.

$ message authentication code (MAC)

*Data associated with an authenticated message allowing a receiver to verify the integrity of the message ([CNSSI-4009]).

$ method

*Single CICM library function that performs a specific task.

$ namespace

*An abstract container that holds related interfaces.

$ non-blocking

*A method is non-blocking if it initiates an operation and then returns control to the caller, usually before the outcome of the operation has been determined. See also blocking.

$ opaque data object

*Binary object accepted by or returned from a method call whose structure is imposed by some entity unrelated to the CICM specification.

$ package

*Software, FPGA image, policy database, configuration parameters, or other types of executable or interpretable code that may be imported into and removed from a module.

$ persistent key

*See static key.

$ policy

*Precise specification of the security rules under which a cryptographic module will operate.

$ port

*Identifier that designates a logical interface through which data moves into and out of a cryptographic module. See also local port and remote port.

$ remote port

*Port in non-local security domain from which transformed data is received. See also local port.

$ role

*A designation to which users are assigned that identifies a job type defined in terms of the privileges of that user.

$ security domain

*System or group of systems operating under a common security policy. Communication between domains is controlled in a well-defined manner.

$ selective bypass

*Portion of the traffic through a channel that is not to be cryptographically transformed. Also commonly referred to as "header bypass."

$ static key

*Cryptographic key imported into or established on a module that will remain on the module until it is explicitly removed. See also ephemeral key.

$ stream

*An abstraction representing an entity utilizing an existing controller to enable data to be sent to a module to be transformed and transformed data to be received using a controller as a foundation.

$ symmetric key

*Usually a sequence of random or pseudorandom bits used initially to set up and periodically change the operations performed in crypto-equipment for the purpose of encrypting or decrypting electronic signals ([CNSSI-4009]). See asymmetric key.

$ system

*Hardware and software components, including the cryptographic module, that meet a specific set of security-related requirements.

$ tamper

*Output signal from module that denotes it has detected a tamper event.

$ token

*See hardware access token.

$ trusted display

*Hardware component independent of a host to enter or display information to be directly sent to/received from a cryptographic module.

$ zeroize

*Input signal instructing the module to clear its memory of any sensitive cryptographic material. CICM supports both a module zeroize (destroying all key material on module) and zeroizing an individual key.

Authors' Addresses

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

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