The Common Interface to Cryptographic Modules (CICM) work is sponsored by the US Air Force Cryptographic Modernization Program, explicitly to provide a common application programming interface for use with high-assurance cryptographic modules in critical settings. This work is also intended to be available for use in other settings, too, such as providing cryptographic services for commercial, proprietary, or personal applications.

**Disclaimer:** CICM is a work in progress. The concepts and details presented here are subject to change.
The Common Interface to Cryptographic Modules (CICM) is intended to support high-assurance cryptographic modules in critical settings, as well as commercial modules used in other settings.

CICM is intended to support

- normal operation of the cryptographic modules,
- management of the cryptographic resources contained in (or controlled by) the modules, and
- management of the module itself.
Current commercial APIs are not a good fit for high-assurance applications, primarily because they do not support separation of security domains (instead, they deliver cryptographic services following a co-processor model).
replaceability, interoperability
A common, logical view of the functions and services facilitates replacing modules with others providing equivalent services.

comprehensive functionality
CICM is intended to support normal operation of the cryptographic modules, management of the cryptographic resources contained in (or controlled by) the modules, and management of the module itself.

assurance
CICM is explicitly designed for use with high-assurance cryptographic modules in critical settings.

performance
Accommodation of “direct” data lines allows CICM to be used in high-performance systems.

standards
CICM itself does not mandate use of standards such as TAMP or CMS, but it supports their use with modules that recognize these standards. (Actually, CICM exchanges these messages with the module but otherwise ignores their content.)

flexibility
CICM is intended to be flexible enough to accommodate future needs and technologies as they arise, although we recognize that this last intent may be a bit difficult to satisfy completely.
As far as CICM is concerned, applications run in security domains that may be connected by cryptographic modules but are otherwise isolated from each other. There may be a number of these domains, but in most cases, no more than two domains will be involved for a normal communication channel through the module (perhaps encrypting the traffic in one direction, and decrypting in the other).

Unlike the co-processor model, this model allows high-performance traffic flow, while reducing opportunity for potentially harmful interactions among the security domains.
Module management includes functions like loading or updating module software and cryptographic algorithms, setting up hardware access token associations and interpreting those tokens, configure interfaces (including any key load interfaces), reporting on cryptographic engines and their states, retrieving module information, administering the module log, and performing module-level time management.

Algorithm and policy management are pretty much just importing or updating these items, and reporting to the application what the module has.

Data services (encrypting or decrypting traffic) take the form of reading from and writing to the appropriate port on the channel. This differs from co-processor models used in some other APIs, and it does not include use of “direct” data lines to the module.

In addition to functions like loading, exporting, generating, or zeroizing keys, key management also includes support for key management protocol sessions. CICM does not mandate which key management protocols to use. Instead, the module itself (or perhaps its supporting libraries) validates the protocol messages and returns generic advice to the application, e.g., to say “error noted” or “protocol session done.”

Channel management includes channel set-up and tear-down, also enable and disable, and channel grouping.

Trust anchor management is concerned with the traditional means by which the module knows which infrastructure material to believe.
These may include key management protocols, module software maintenance, algorithm distribution, …

The protocol session fragment shown in the example above is between the cryptographic module and another participant, in this case a certification authority (the other participant could as easily have been a key distribution center).

CICM plays at most a supporting role in one of these protocol sessions. In such a case,

The application uses CICM to convey messages to or from the module.
CICM itself does *not* interpret the messages; it just considers them to be blocks of binary data.
The module

  Validates the messages;
  Interprets the messages, thus executing the protocol;
  Generates the appropriate responding messages and returns them to the application; and
  Tells the application about progress, *e.g.*, “error noted” or “awaiting next message”

The application forwards the responses to the other participants.
The module will do it

The approach we take to cryptographic material and management protocols recognizes that the module will enforce policy and implementation constraints. CICM just exchanges messages with the module, depending on the module to validate and understand the messages, and then report whatever the application needs to know to go on to the next step.

CICM does this for importing policy specifications, cryptographic algorithms, module software, key material, certificates, revocation lists, trust anchors, …

Apart from conducting exchanges with the module, CICM does not interpret this information
Current Status

♦ **Interface development effort**
  – API draft specification version 0.4 released last month (limited availability)
  – Prior industry review completed
  – Specification development effort ongoing
  – Prototype efforts to validate concept ongoing

♦ **Next steps**
  – Publish DRAFT version 0.5 in near future, with *no* release restrictions
  – Support and enable adoption for both commercial and high-assurance service
  – Seeking move to a standards organization

“Securing the Global Information Grid (GIG)”
A Few Questions …

- What should we watch out for?
- What weaknesses or missed opportunities are we risking?
- What approaches, techniques, tools, or formalisms should we consider?

We welcome all the advice or help you can offer
Dan is leading this work, and Sam is a techie on the project.
Backup slides follow

“Securing the Global Information Grid (GIG)”
Despite the earlier “partitioned” model, which shows multiple domains (two, actually), the CICM view for a given application is actually limited to
the application itself,
the CICM instance that is shared by the application and the cryptographic subsystem, and
the cryptographic subsystem, which includes some components residing on the host system and includes the cryptographic module itself.

This view is concerned with only one side only (\textit{e.g.}, red or black), and in general, CICM does not care which side it is.

The host system will likely have multiple applications running at the same time, each with its own instance of CICM providing access to the cryptographic subsystem.
Local Identifiers

- Designate keys, modules, algorithms, channels, channel service bindings, ...

- Never reused, for limited values of “never”
  - Significance limited to generating process (thus “local”)
  - Significance limited to the kind of entity it designates (key id 4 differs from channel id 4)

- Postulate that process will terminate before identifiers for a given type are all used up (upon “wrap-around,” refuse to generate any more of these identifiers for this process)
Data Service Constraints

- Port serves only one channel at a time

- Port provides data service (send, receive traffic) to only one application process at a time
  - “Channel service identifier” designates binding of port, channel, and traffic flow direction(s)
  - Send, receive both require channel service identifier

  - No explicit mechanism (yet) to release the port binding, which would allow another process to take over the port as it serves the current channel

“Securing the Global Information Grid (GIG)”