A Comparison of CDSA to Cryptoki

Ruth Taylor
rct@epoch.ncsc.mil
National Security Agency

Agenda

What is a CAPI?

Motivation for Study

CDSA and Cryptoki Overviews

Mapping the Calls: Trends

Differences in Auxiliary Functionality

Alternate Implementations

Porting Issues

Adaptation Layers

What is a CAPI?

CAPI: Cryptographic Application Programmer Interface

- Set of calls allowing an application to access cryptography
- High level: programmer requests “encrypt”
- Low level: programmer requests “encrypt” with
  3DES, CBC mode, IV=x, and y padding bytes
- Both CDSA and Cryptoki are low level CAPIs

Why use a CAPI?

- Cryptographic library correctness and reusability
- Modularity => isolates application or cryptography modifications
- Isolation of cryptography => better system security
  - Cryptography in a separate library: good
  - Cryptography in a separate process: even better
  - Secure OS is needed for true security

Motivation for Study

Which CAPI to use in Flask?

- Flask: secure OS prototype jointly developed by the University of Utah and R23's Trusted Operating Systems Team

Useful to CAPI Community

- Contrast two common CAPIs
- Help with ports & layering between CDSA and Cryptoki

CDSA Overview

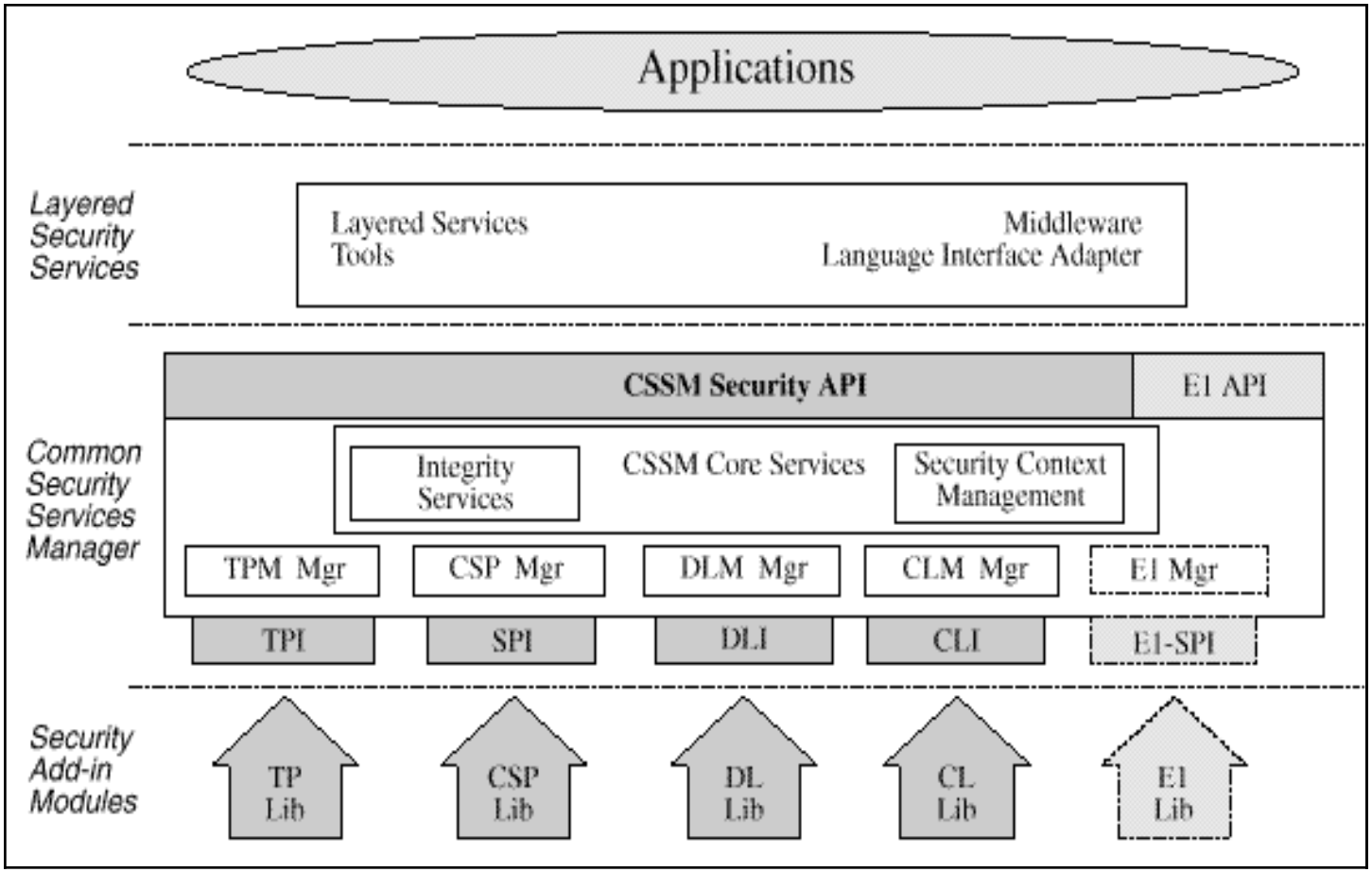
CDSA: Common Data Security Architecture

- Originally developed by Intel Architecture Labs (1997)
- Standardized by Open Group (December 1997)
- Support from IBM, Netscape, Entrust, TIS, Sun, HP, Motorola, Shell, J.P. Morgan...

Features

- Layered architecture:
  - API to crypto, trust policy, data storage, certificate services
  - Security administrator add security modules of their choice
- Different interfaces for applications and security service libraries
- Rich in auxiliary functionality
  - verifies authenticity & integrity of add-in modules, registers module capabilities, caches user security contexts
  - memory management for applications

CDSA Overview*



TP=Trust Policy, CSP=Cryptographic Service Provider, DL=Data Library, CL=Certificate Library, EL=Elective Library

*Figure extracted from Open Group CDSA Specifications c707, ISBN 1-85912-194-2.

Cryptoki Overview

Cryptoki = RSA's PKCS #11

- A CAPI to cryptography services ONLY
- single interface for both applications & crypto libraries
- simpler than CDSA => less auxiliary services provided
  - no memory management for applications
- originally designed for direct hardware interfacing
  - can also handle software tokens

Terminology

- mechanism: a cryptographic algorithm
- token: module which implements the mechanism
- slot: abstract adaptor which holds a token
- session: logical connection between an application and token
- objects: specific data types (e.g. DES3 secret key)

Quick Comparison: CDSA vs. Cryptoki

Similarity:

- Both define low-level interfaces to cryptography

Difference: Auxiliary Services

- CDSA is an *architecture* with cryptographic, data storage, certificate, and trust policy libraries
- Cryptoki only defines the cryptographic interface
- Therefore, CDSA is richer in auxiliary services to manage the architecture (module management, memory management)

Difference: Hardware Interface

- CDSA: uses either hardware or software cryptography
- Cryptoki: originally designed to interface to hardware, and has more direct interfaces to hardware cryptographic tokens

CDSA Contexts v. Cryptoki Objects

CDSA Contexts:

- Package containing all information needed to perform the call: algorithm, key data, initialization vector, mode, padding, valid dates...

Cryptoki objects:

- Define the exact format for a specific data type (secret key, certificate) and algorithm
- E.g., DES3 secret key object or RSA public key object

Cryptoki mechanisms + objects ~~ CDSA contexts

- Objects/contexts are created, manipulated, or destroyed beneath the API, which returns a handle. Application can't directly manipulate them.
- Object and context handles are passed as parameters to CSPs

Mapping the Calls: Trends

(CSSM API & SPI Mapped to Cryptoki calls)

- Cryptographic calls: mapped 1-to-1, 88% equivalence; those not equivalent had alternate implementation methods
- Calls to auxiliary functionality: many differences
- CDSA was higher-level than Cryptoki
  - `CSSM_GetModuleInfo` -> `C_GetSessionInfo, C_GetSlotList, C_GetSlotInfo, C_GetTokenInfo, C_GetMechanismInfo, C_GetInfo`
  - Cryptographic calls:

```
C_*Init(mechanism_ptr,sesssion_handle,objects,data);
C_*;

CSSM_Encrypt(context, data);
```

- See paper for category mappings & technote for call mappings

Unique Auxiliary Functionality: Cryptoki

Token Administration

- Cryptoki has a simple token administration model
  - Security officer must initialize token
  - Users must login w/ PIN before accessing private objects
- `C_InitToken`: Initializes token by destroying objects and denying normal user access until PIN is set
- `C_InitPIN`: Initializes a normal user's PIN

Low-level Hardware Interface

- `C_WaitForSlotEvent`: Wait for slot event (eg. token insertion)
- `C_InitToken, C_InitPIN, C_SetPIN, C_Login`: Allow user to interface w/ a protected authentication path on token (e.g. PINPad)
- `C_CloseAllSessions`: Optionally ejects token

Unique Auxiliary Functionality: CDSA

Module Administration

- `CSSM_Module(Un)Install`: Add/delete module from CSSM Registry
- `CSSM_VerifyComponents`: Authenticate CSSM components and verify their integrity
- `CSSM_Get{Handle,GUID}Usage`: Return bitmask describing module's services given a handle or GUID
- `CSSM_SetModuleInfo`: Set module description information
- `CSSM_ListModules`: List all currently registered modules
- `CSSM_FreeInfo, CSSM_Freelist, CSSM_Free, CSSM_FreeKey, CSSM_FreeModuleInfo, CSSM_GetAPIMemoryFunctions`: The CDSA memory management functions

Unique Auxiliary Functionality: CDSA

Different Architectures

- `CSSM_GetInfo`: Returns version information for CSSM instances
- `CSSM_RetrieveCounter`: Returns value for a tamperproof clock
- `CSSM_VerifyDevice`: Force a cryptomodule to do self-verification and integrity testing
- `CSSM_RequestCSSMExemption`: Application requests exemption from a CSSM standard built-in security check

Alternate Implementations: Cryptoki

- `C_SignRecover`: Signs data such that the data can be recovered.
  -> call `CSSM_EncryptData` with a private key
- `C_VerifyRecover`: Verifies signature and recovers data.
  -> call `CSSM_DecryptData` with a private key
- `C_CloseAllSessions`: Close all sessions between application and token, destroying all session objects.
  -> call `CSSM_ModuleDetach` repeatedly; when last session is closed, objects should be removed.
- `C_InitPIN`: Initializes a normal user's PIN through the Cryptoki software or using a protected authentication path (PINPad).
  -> CDSA allows all security administration tasks to be defined as `CSSM_PassThrough` functions.

Alternate Implementations: CDSA

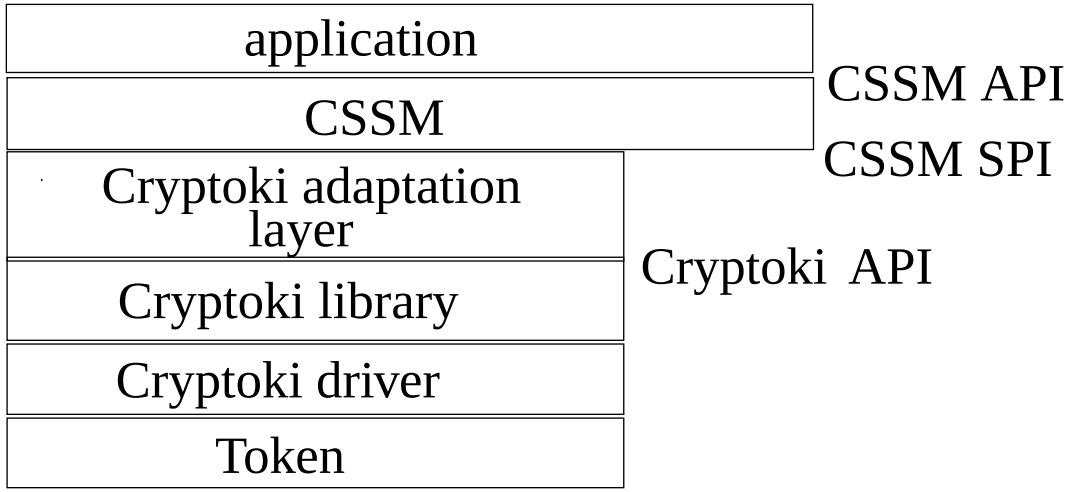
- `CSSM_VerifyComponents`: Verifies CSSM components
  -> Cryptoki tokens can be authenticated w/ built-in certificates or tokens may be challenged to sign a message w/ secret key.
- `CSSM_QuerySize`: Returns size of output buffer needed.
  -> In Cryptoki, call the cryptographic fxn w/ a NULL buffer.
- `CSSM_{Generate,Verify}MAC`: Message Authentication Code
  -> In Cryptoki, use the signature operations w/ symmetric keys
- `CSSM_PassThrough`: Allows an application to access functionality implemented by the CSP but not part of the CSSM API.
  -> Extend Cryptoki's interface by simply adding calls.

Porting Issues (Summary)

Need to consider..

- Converting function names, parameter lists, call sequences
- Memory Management
  - CSSM API optionally does this for applications
  - Cryptoki requires application to do
- CSP Verification
  - CSSM does CSP verification and built-in security checks
  - In Cryptoki, these are implemented independent of the API
- Interfaces to Hardware CSPs
  - Cryptoki provides interfaces to wait for token removal or login through protected authentication paths
  - In CDSA, these must be implemented as a PassThrough fxn

Adaptation Layer



- 1998: Intel demo'd a CDSA over Cryptoki Adaptation Layer
- Advantages of this approach:
  - Cryptoki is lower level, and better built upon
  - A Cryptoki library, with an adaptation layer mapping it to the CSSM SPI, fits into the CDSA model as a CSP

Questions?

Ruth Taylor
rct@epoch.ncsc.mil