Common Data Security Architecture (CDSA) Formal Development

Peter White

Galois Connections

Outline

- What is CDSA?
- Why formal development of CDSA?
- Our CSSM architecture
- Our CDSA demo
- Future of CDSA Formal development

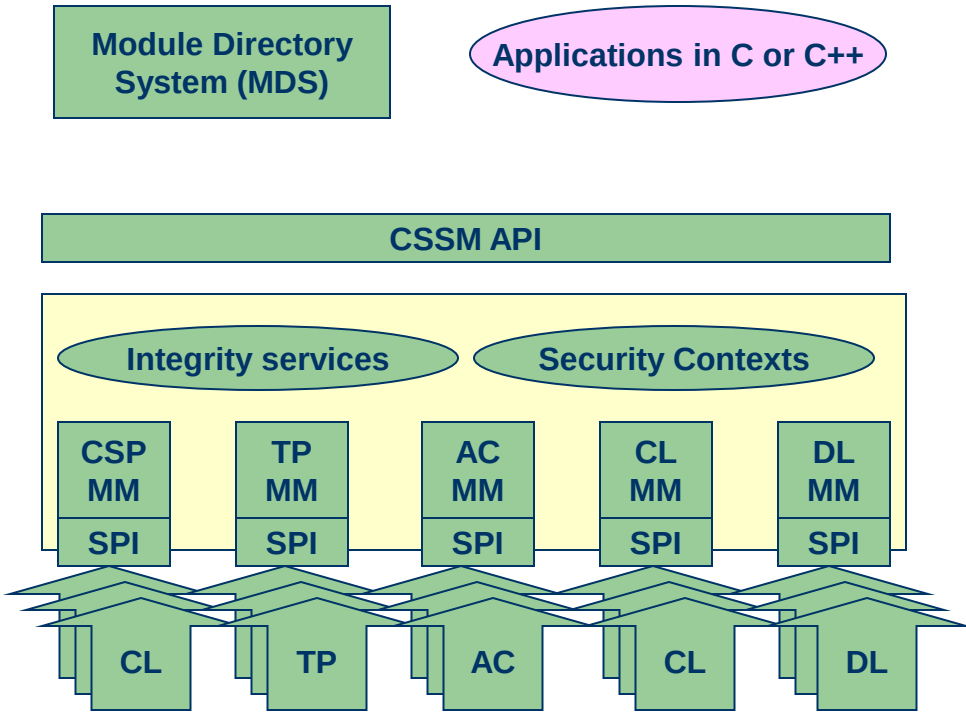
What is CDSA?

A place to plug my pluggable crypto

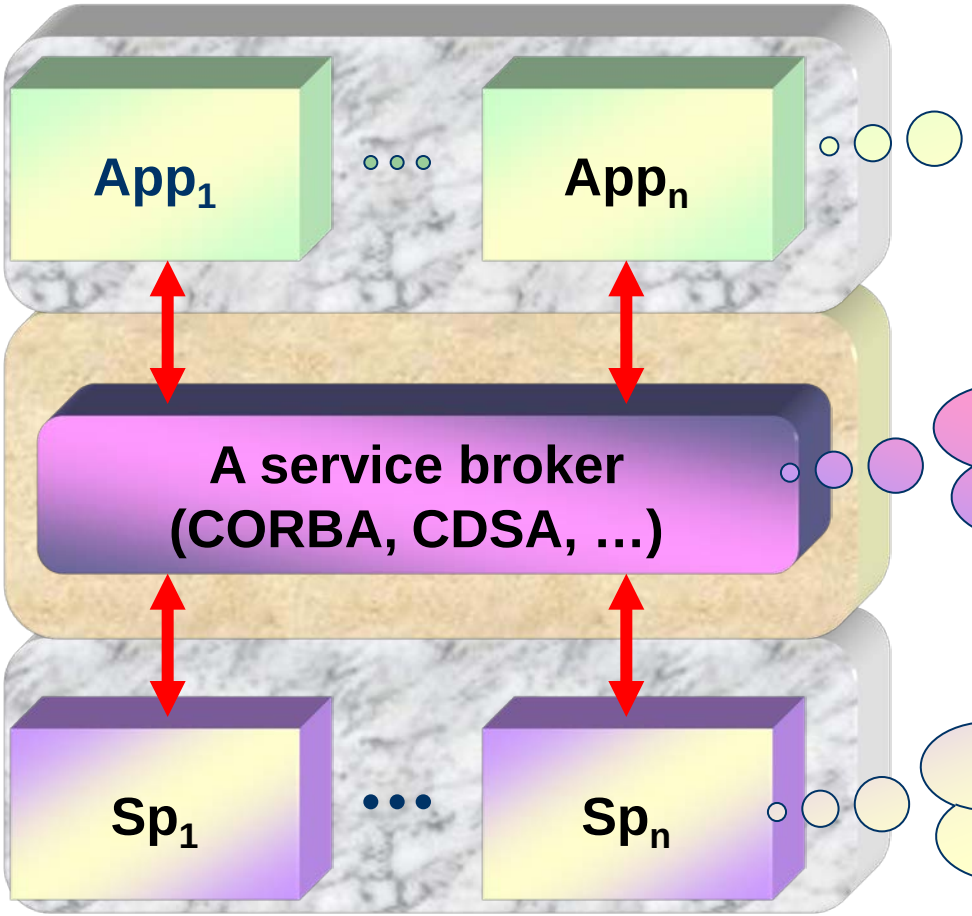
```
Legend
AC   = Authorization Computation
CL   = Certificate Library
CSSM = Common Security Service Manager
CSP  = Cryptographic Service Provider
DL   = Data Library
MM   = Module Manager
SPI  = Service Provider Interface
TP   = Trust Policy
```

A receptacle for pluggable crypto

- Provide common API to pluggable CSPs and CLs
  - Common Application API
  - Common SPI
  - Underlying service can be hardware, software or both
  - Maintain algorithm context
- Provide assurance of the integrity of underlying modules
- Provide a multi-threaded execution environment



The business case for CDSA: CDSA as a market maker

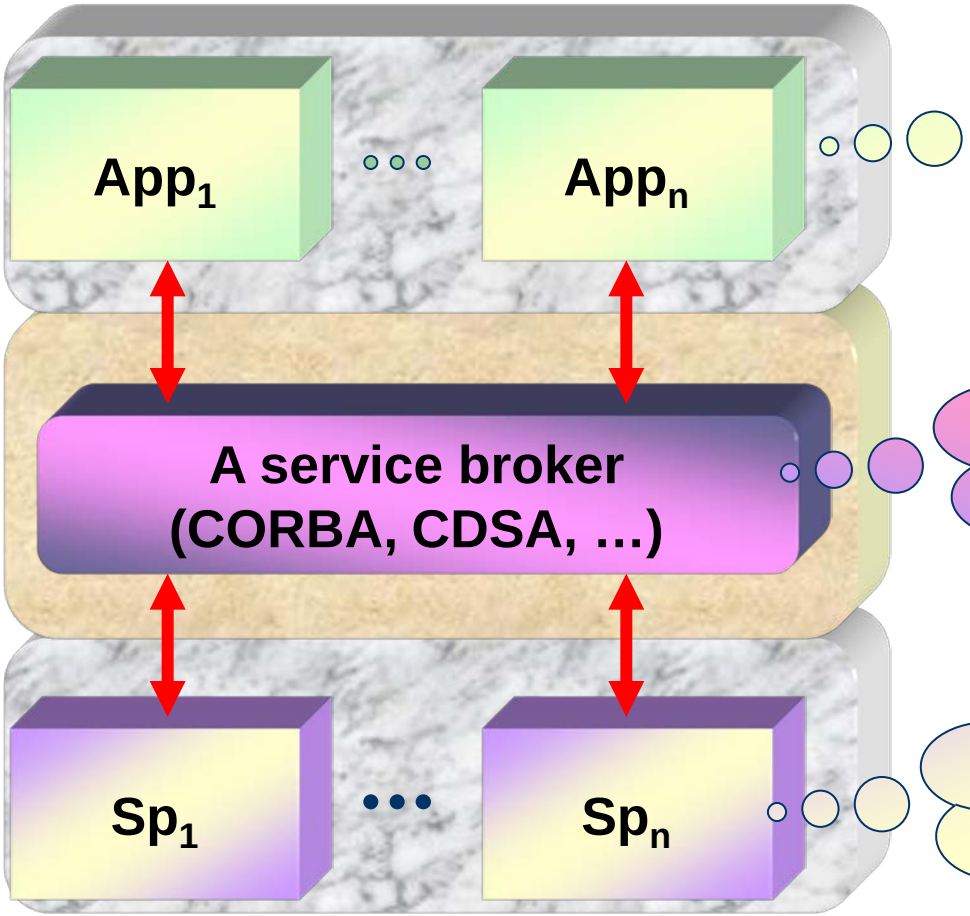


The application provider seeks as large a market as possible, selling to users equipped with a variety of underlying service technologies.

The broker mediates the desires of the application provider and the service provider, by standardizing the interfaces, and providing a platform supporting many service provider plug ins

The service provider also seeks as large a market as possible, servicing users of a variety of applications.

The business case for CDSA:
Avoid commitment until the last minute



Application relieved of many security requirements. Avoids committing to any particular cryptographic technology

Provides flexible market that enables applications and service providers to avoid commitment

The service provider avoids exclusive commitments to particular applications

Why formal development of CDSA?

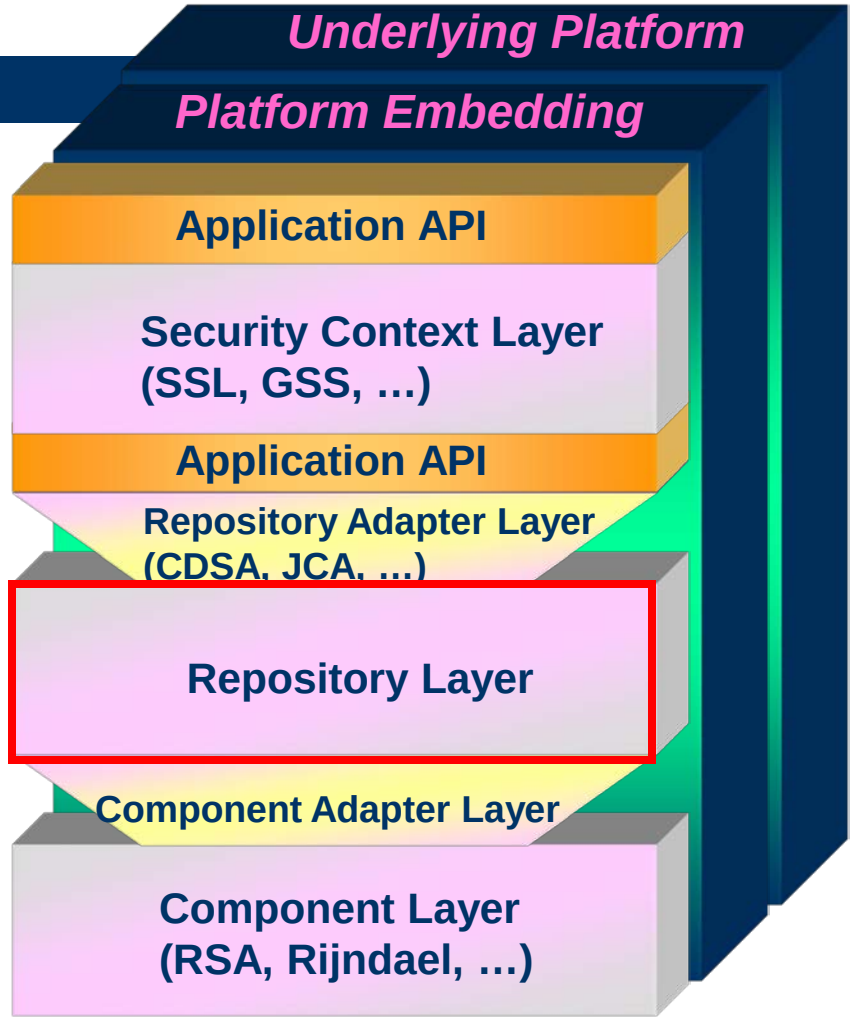
Discovering the essence of CDSA

CDSA and assurance

- Any CDSA can increase assurance of some security properties
- CDSA itself is a low assurance program
  - 1000 page specification
  - 90 megabyte distribution
  - 100,000's C LOC

Achieving high assurance

- Formalize the essence of CDSA
  - Secure Object Repository
- Formalize the properties to be assured
- Implement the essence of CDSA
- Grow to the full CDSA implementation
- Drive the process with an application



Our CSSM (Secure Object Repository) architecture

Highlights of module management

- Module Directory
  - Database of information about modules available on the platform
- Introduce a module
  - Tell CSSM about a module
- Load a module
  - Make the module runnable
- Attach a module
  - Provide a handle to an execution environment for the module
- Create a context
  - Create a context for executing an instance of the module
- Use a context
  - Call an algorithm provided by the module

Architectural theme: Separation of concerns

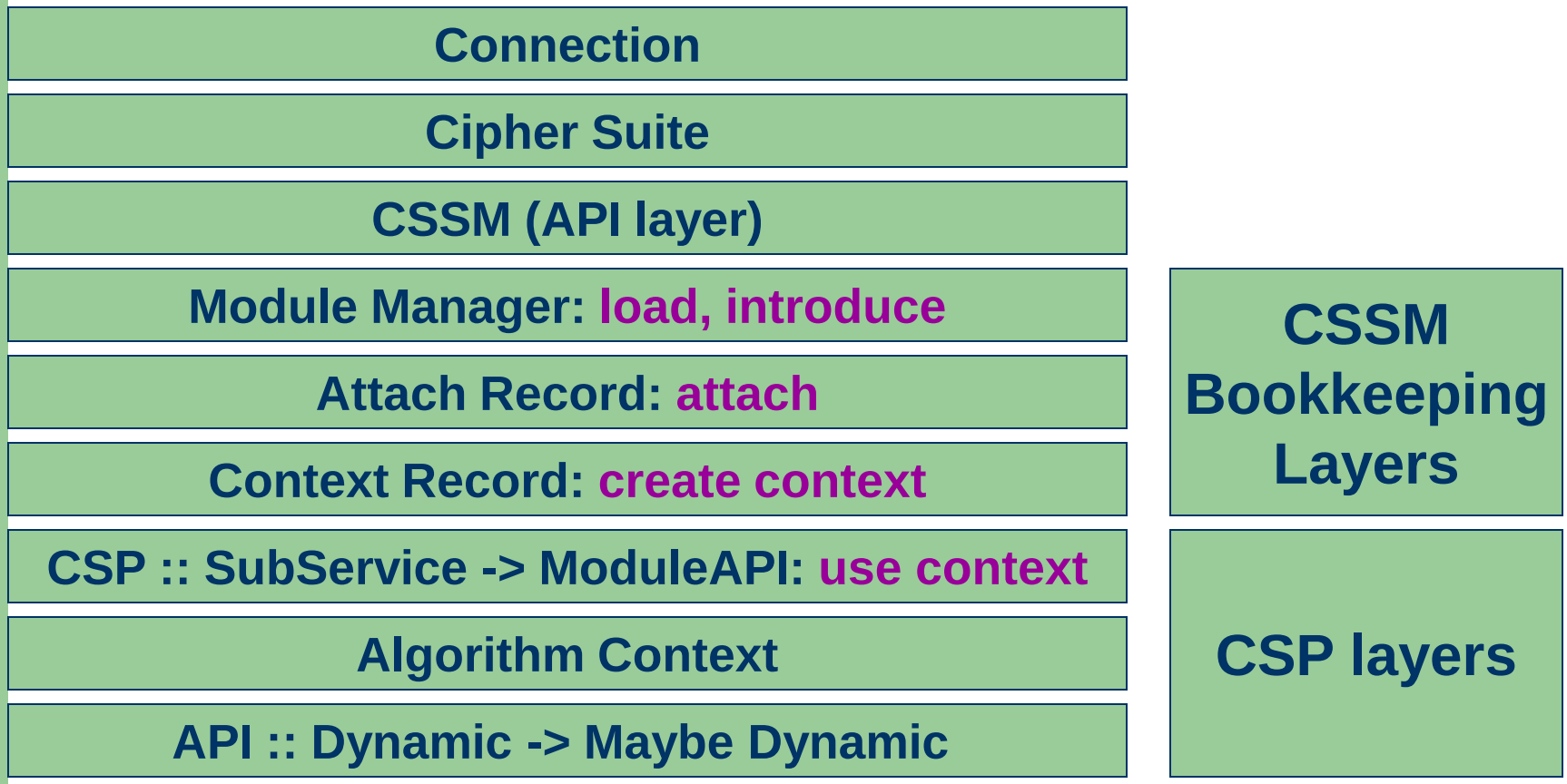
- Memory management
- Protection mechanisms
- Bilateral authentication mechanisms
- Algorithm Context methods
- Plug in receptacle methods
- Authorization mechanisms

Create a structure in which these variables can be isolated, and set to values appropriate to the platform architecture where CSSM is running.

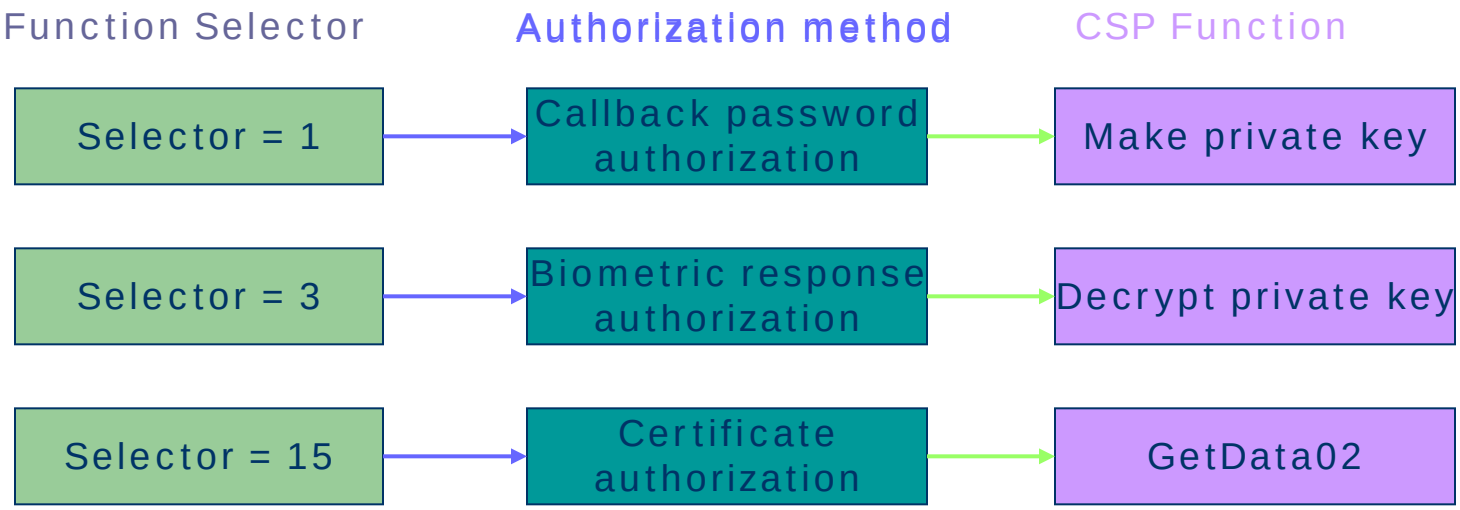
Architectural parameters

- Memory management
  - Application centric
  - Haskell heap
  - Underlying platform
  - CSP centric
- Protection mechanisms
  - None
  - Pointer validation checks
  - Types and parametricity
- Bilateral Authentication
  - None
  - Signed manifests
  - Underlying Platform (Separation kernel)
- Algorithm context
  - CSP centric
  - CSSM centric
- Plug in receptacle
  - Haskell CSP
  - process
  - thread
  - process group
- Authorization mechanism
  - protocol
  - callbacks
  - higher order functions

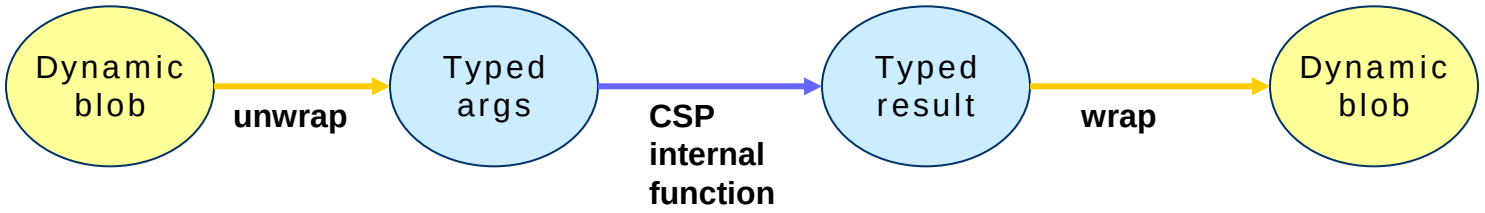
The layers of the architecture



Example of abstraction: The module API



Passing the arguments to the algorithm through CSSM

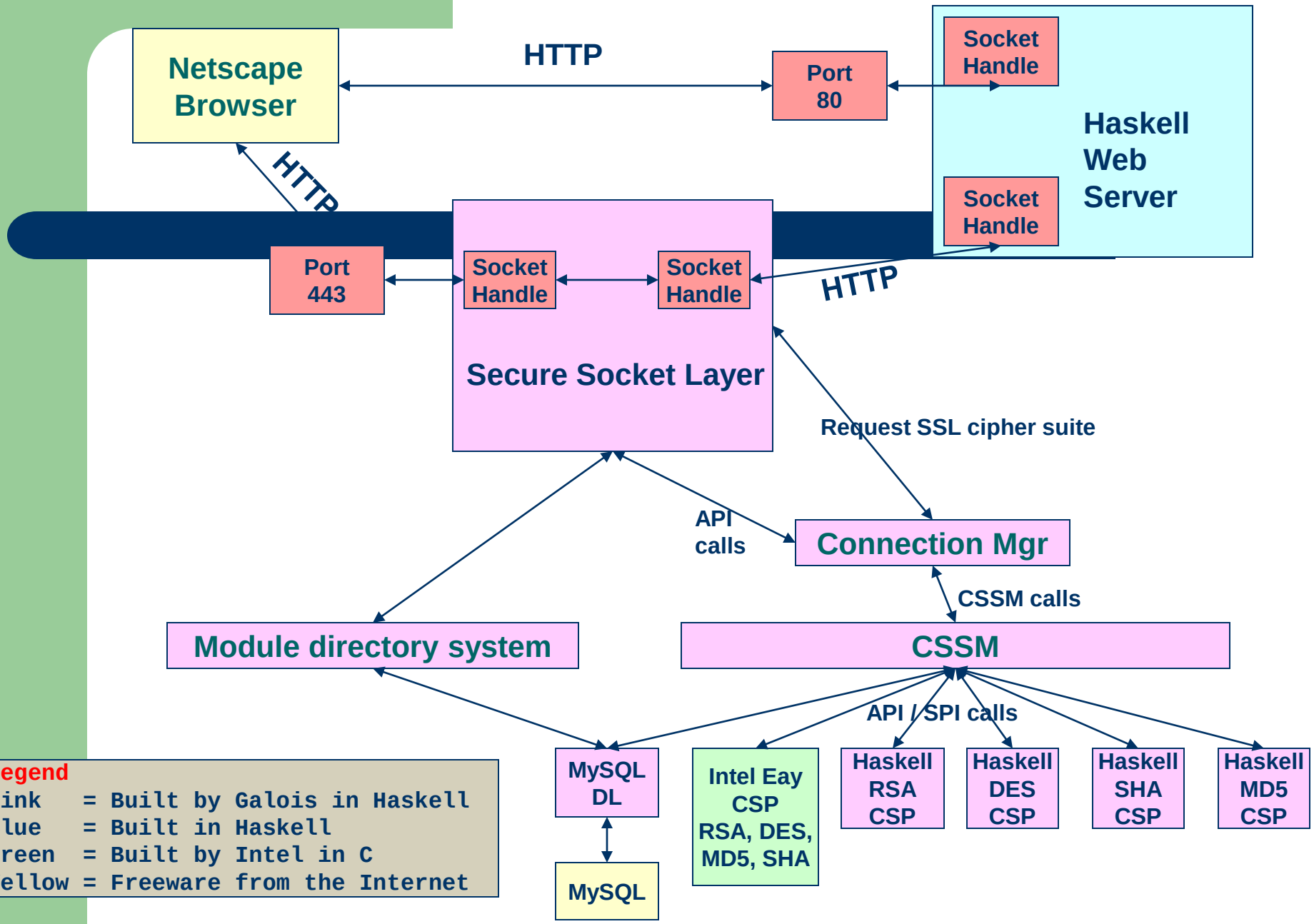


The module API: A Haskell code fragment

```
type APIIO' = ContextParameter -> Dynamic -> IO (Maybe Dynamic)
data API    = API APIIO'
-- // Existential type to hide the details of the authorization
data AuthorizedAPI = forall a. (Authorize a) => AuthorizedAPI a API
-- // Associate module API (and authorization) with function selector
type ModuleAPI = AssocList FunctionSelector AuthorizedAPI
-- // Implement the calling mechanism to a CSP api
apiCall :: Dispatcher -- // items in blue came from calling app
apiCall fs modapi args ctxtp appevidence =
    let mapi = allookup modapi fs –- // find the api for the selector given
    in case mapi of
       Nothing ->
            do { putStrLn ("*** No function with selector " ++ show fs)
               ; return Nothing
               }
       Just (AuthorizedAPI auth (API api)) ->
            if authorize auth appevidence -- //check if authorized
            then api ctxtp args
            else return Nothing
```

Our CDSA Demo

Demo Architecture



Legend

Pink = Built by Galois in Haskell
Blue = Built in Haskell
Green = Built by Intel in C
Yellow = Freeware from the Internet

Future of CDSA Formal Development

Next tasks

- Add CL component
  - Very easy to do in Haskell, except
  - Requires ASN.1 / BER / DER encode and decode to be developed in Haskell
    - To be implemented using parser combinators
- Export some of CSSM and prove some properties, for example
  - Module can be attached only if loaded
  - API can be called only if authorized

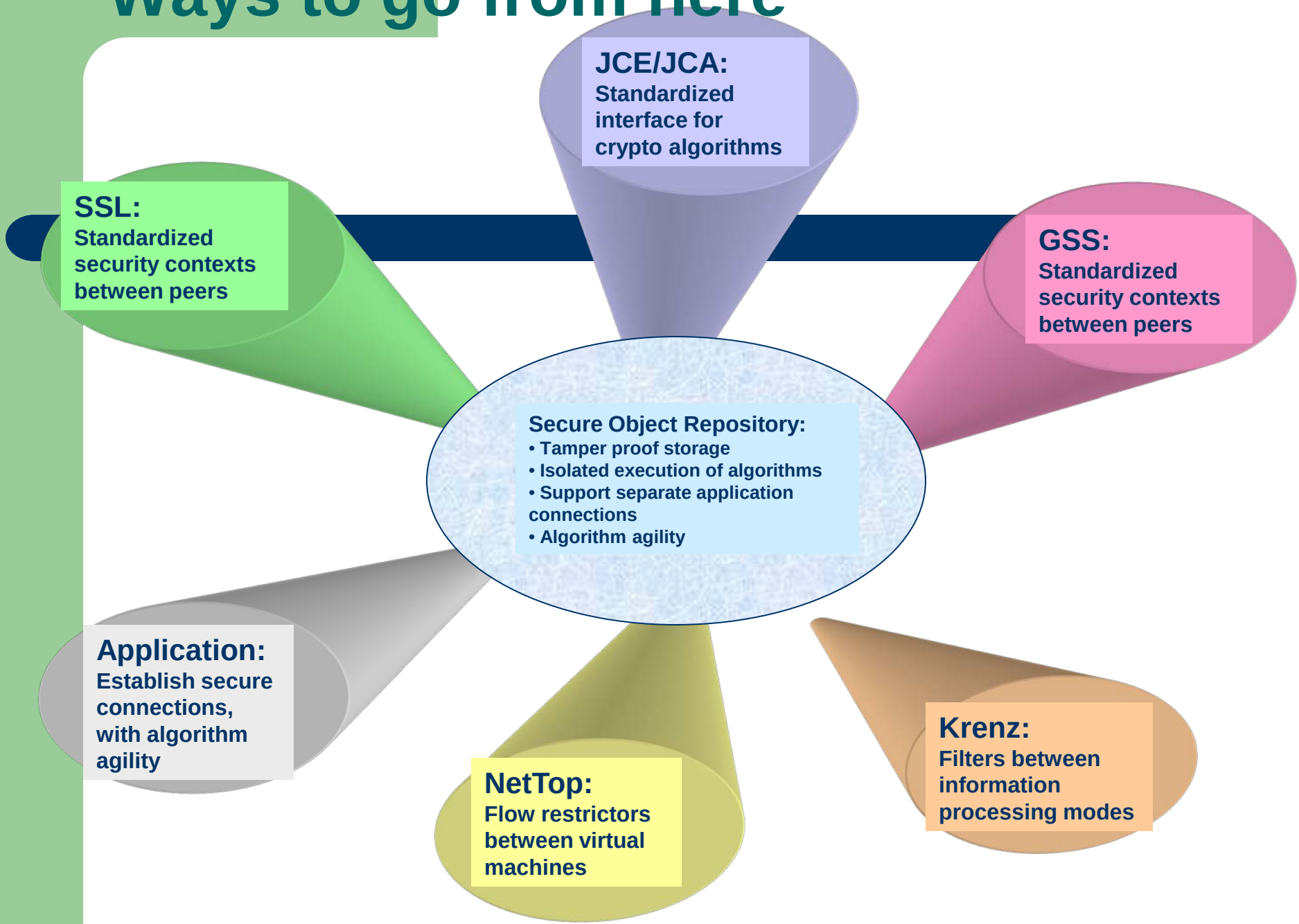
Future tasks
CDSA Trust Policy Description

- CDSA Specification defines a very nice concept of trust policy in terms of a graph of credentials, actions, and objects
  - Does the requestor have sufficient credentials to perform the requested action upon the specified object?
- Translate this description into an abstract graph data structure, and generalize the “trust” relationship so that different trust evaluation methods can be used.
- Implement this in a trust policy module

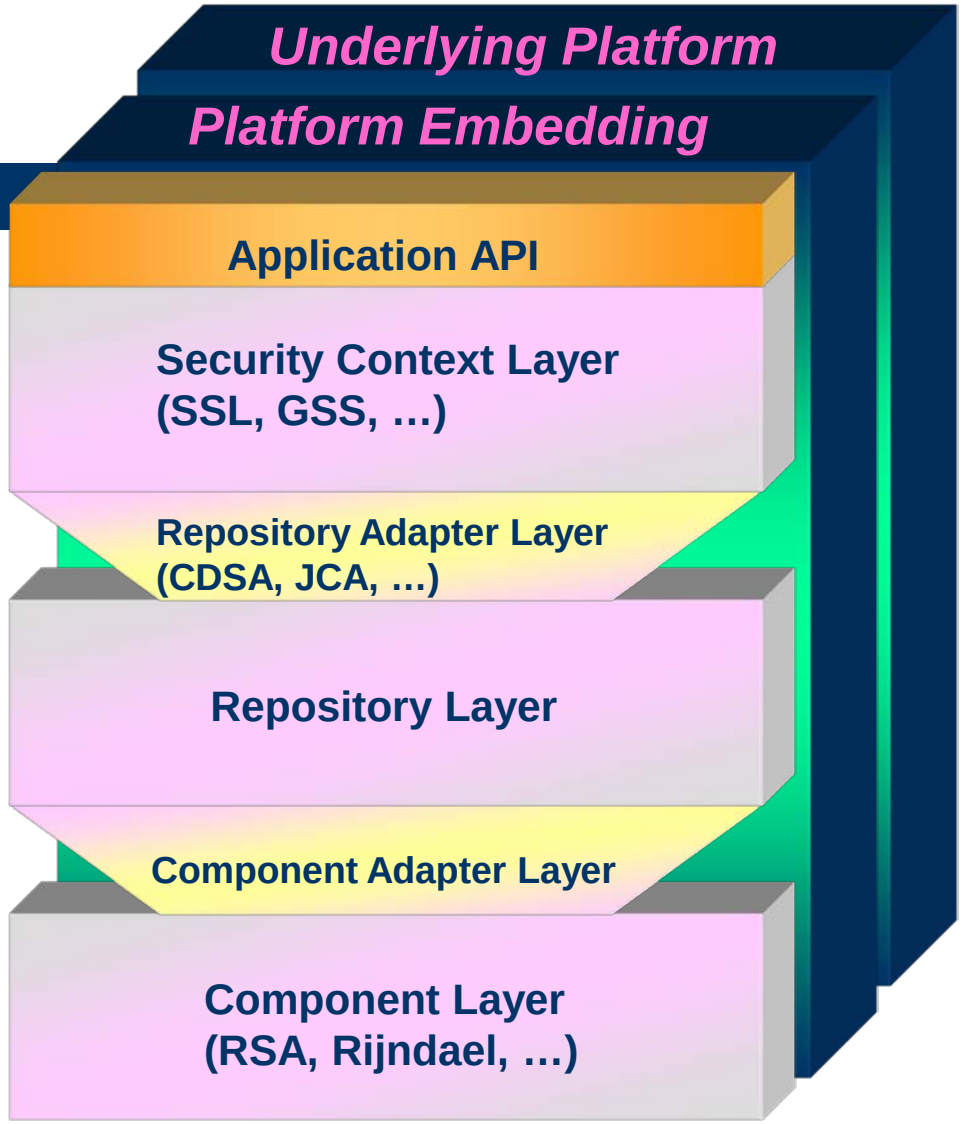
Future tasks

- Bilateral Authentication
  - Ensuring the application that the CSP being used is the CSP that is desired
- Multi threading
  - Permitting concurrent operation of CSPs, in support of multiple applications
  - Haskell has very powerful thread primitives

Ways to go from here



The vision


Underlying Platform
Platform Embedding
Application API
Security Context Layer
(SSL, GSS, …)
Repository Adapter Layer
(CDSA, JCA, …)
Repository Layer
Component Adapter Layer
Component Layer
(RSA, Rijndael, …)