The MILS Component Integration Approach To Secure Information Sharing

Carolyn Boettcher, Raytheon, El Segundo CA

Rance DeLong, LynuxWorks, San Jose CA

John Rushby, SRI International, Menlo Park CA

Wilmar Sifre, AFRL/RITB, Rome NY

MILS

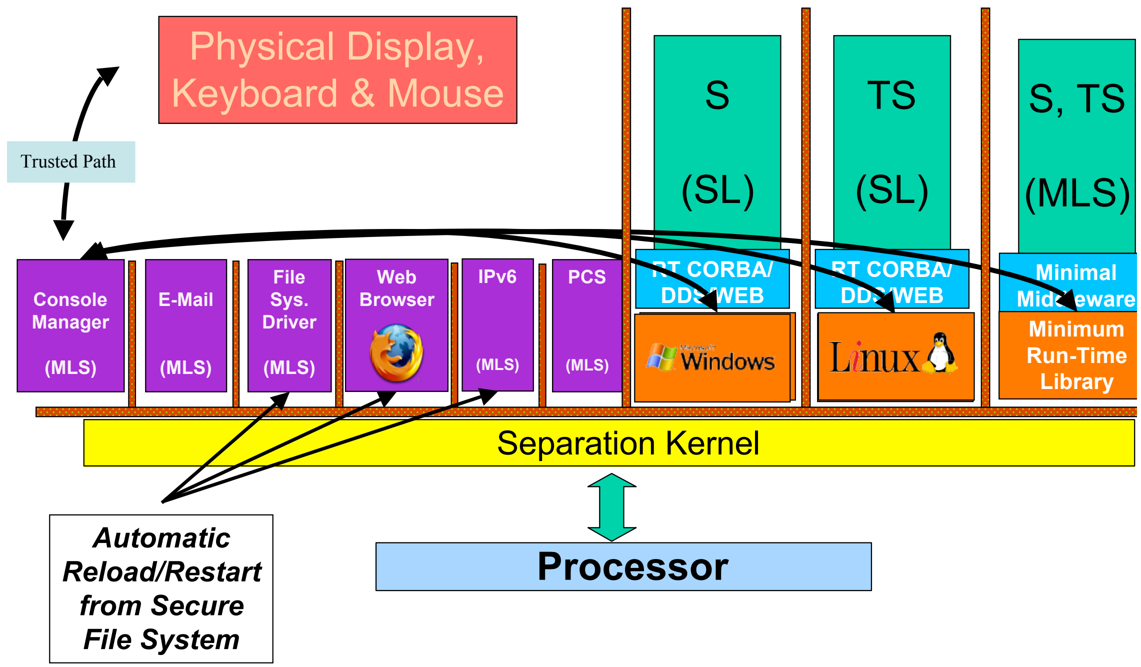
- Is a security architecture adopted for
  - F22, F35, FCS, JTRS, DDG-1000, CDS

  among others
- We are talking about security as a critical property
- So need to provide strong assurance that it is achieved
- We build systems from components
- And we'd like critical properties and assurance to compose component-wise as well
- That's the topic of this talk
- I also want to persuade you the approach might work for safety (i.e., IMA) as well as security
- And for enterprise (e.g., ground) and commercial systems, as well as embedded

The MILS Idea

Traditionally presented as three layers

- Separation kernel, middleware, applications



Somewhat similar to IMA

The MILS Idea (ctd)

- Problem is, that doesn't compose
  - i.e., it's not clear how you get a certified MILS system out of certified MILS components and subsystems
  - Without opening everything up
  - IMA has a similar problem

- I'll present a MILS Component Security Integration approach based on two levels

- That is compositional

Component Security Integration

- We build systems from components
- And we'd like security properties and assurance/certification to compose
  - That is, assurance for the whole is built on assurance for the components
- Seldom happens: assurance dives into everything
- The system security assurance argument may not decompose on architectural lines (Ibrahim Habli & Tim Kelly)
  - So what is architecture?
  - A good one simplifies the assurance case

The MILS Idea (Two-Level Version)

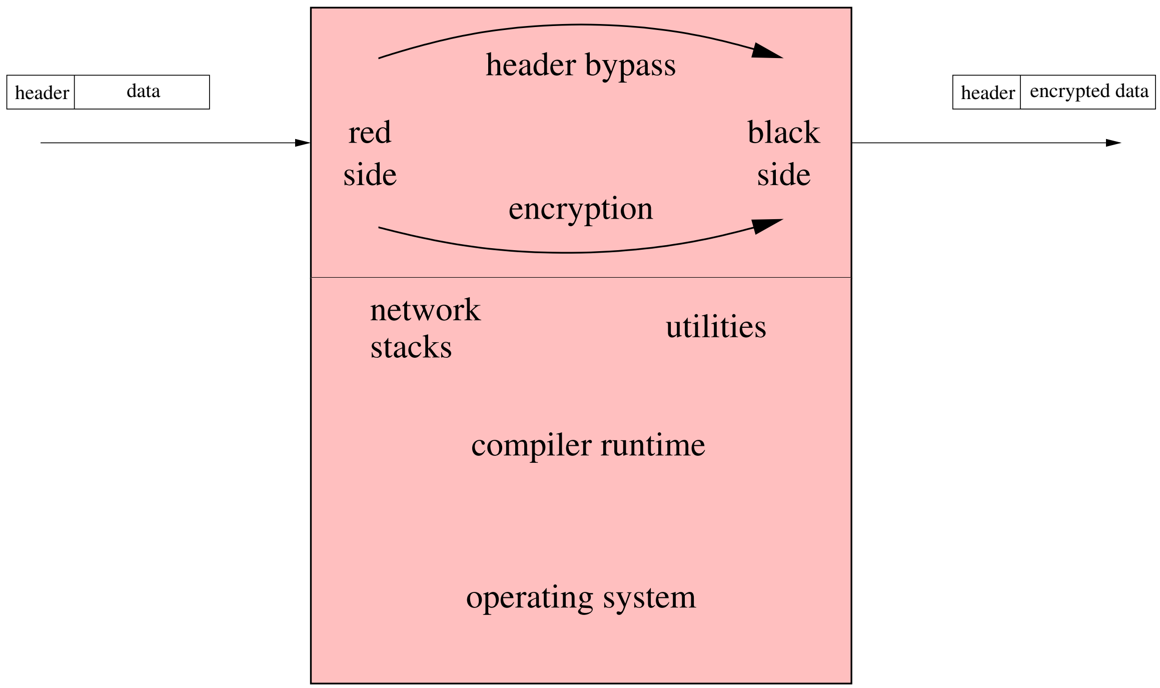
- Construct an architecture so that security assurance does decompose along structural lines
- Two issues in security:
  - Enforce the security policy
  - Manage shared resources securely
- The MILS idea is to handle these separately
- Focus the system architecture on simplifying the argument that policy is enforced correctly
  - Hence policy architecture
- The policy architecture becomes the interface between the two issues

Policy Architecture

- Intuitively, a boxes and arrows diagram
    - There is a formal model for this
- Boxes encapsulate data, information, control
    - Access only local state, incoming communications
    - i.e., they are state machines
- Arrows are channels for information flow
    - Strictly unidirectional
    - Absence of arrows is often crucial
- Some boxes are trusted to enforce local security policies
- Want the trusted boxes to be as simple as possible
- Decompose the policy architecture to achieve this
- Assume boxes and arrows are free

Crypto Controller Example: Step 1

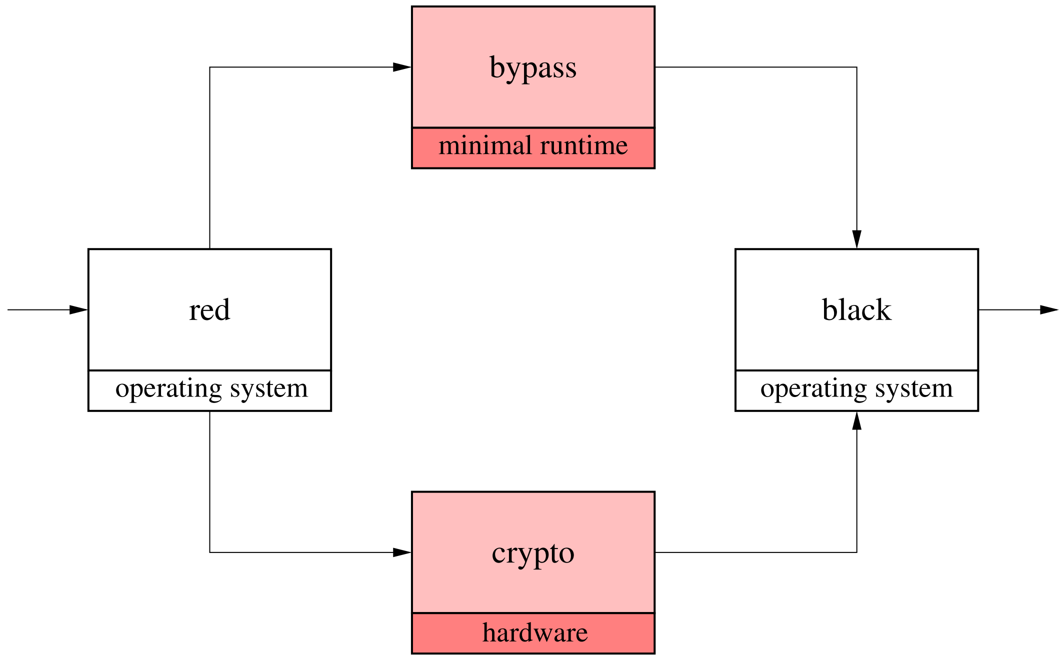
Policy: no plaintext on black network



No architecture, everything trusted

Crypto Controller Example: Step 2

Good policy architecture: fewer things trusted



Local policies (notice these are intransitive):

Header bypass: low bandwidth, data looks like headers

Crypto: all output encrypted

Policy Architecture: Compositional Assurance

- Construct assurance for each trusted component individually
  - i.e., each component enforces its local policy
- Then provide an argument that the local policies
  - In the context of the policy architecture

  Combine to achieve the overall system policy
- Medium robustness: this is done informally
- High robustness: this is done formally
  - Compositional verification

Compositional Verification for Policy Integration

- Need to specify what it means for a component to satisfy a policy under assumptions about its environment
- Then show how these compose (policy of one component becomes the assumptions of anther)
- Fairly standard Computer Science
- MILS is agnostic on the exact approach used
    - Policies/assumptions as properties
    - Or as abstract components

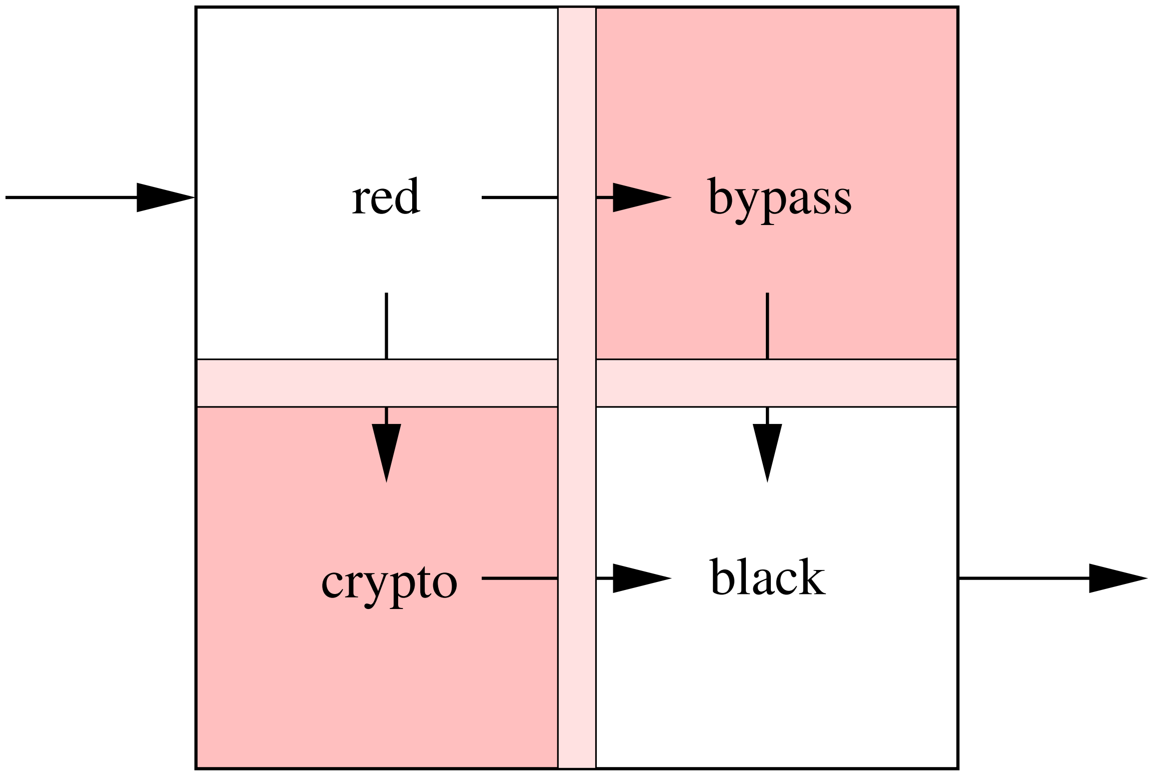
Resource Sharing

- Next, we need to implement the logical components and the communications of the policy architecture in an affordable manner

- Allow different components and communications to share resources

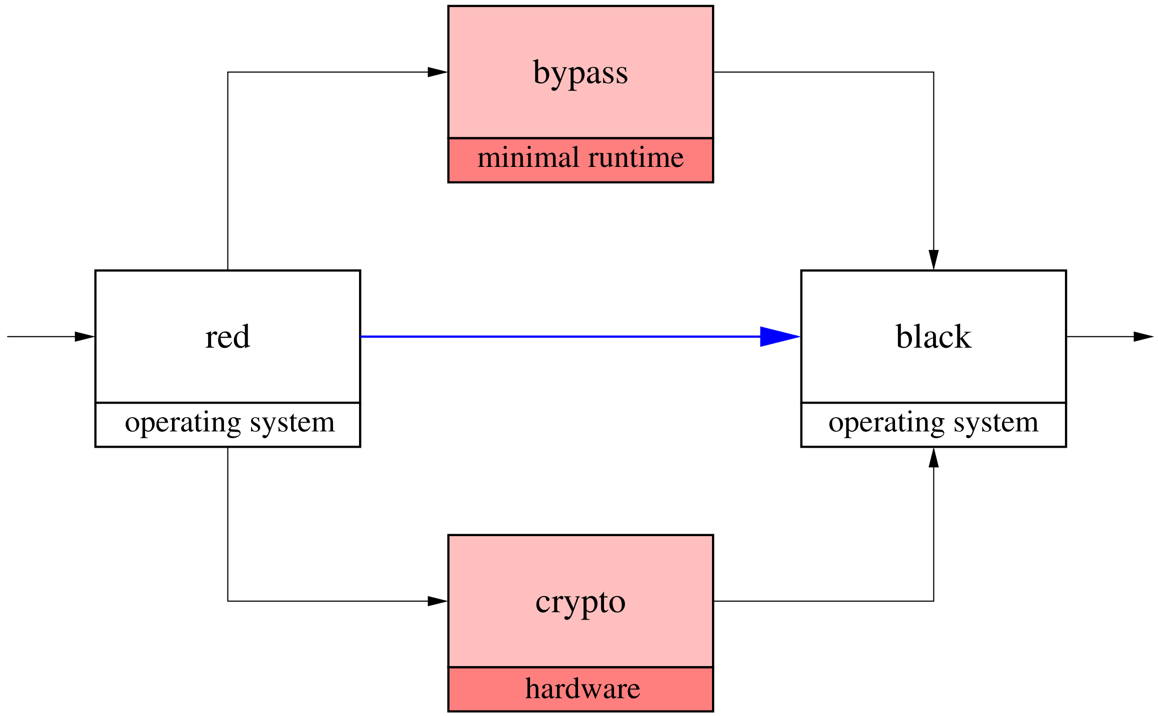
- Need to be sure the sharing does not violate the policy architecture
  - Flaws might add new communications paths
  - Might blur the separation between components

Poorly Controlled Resource Sharing

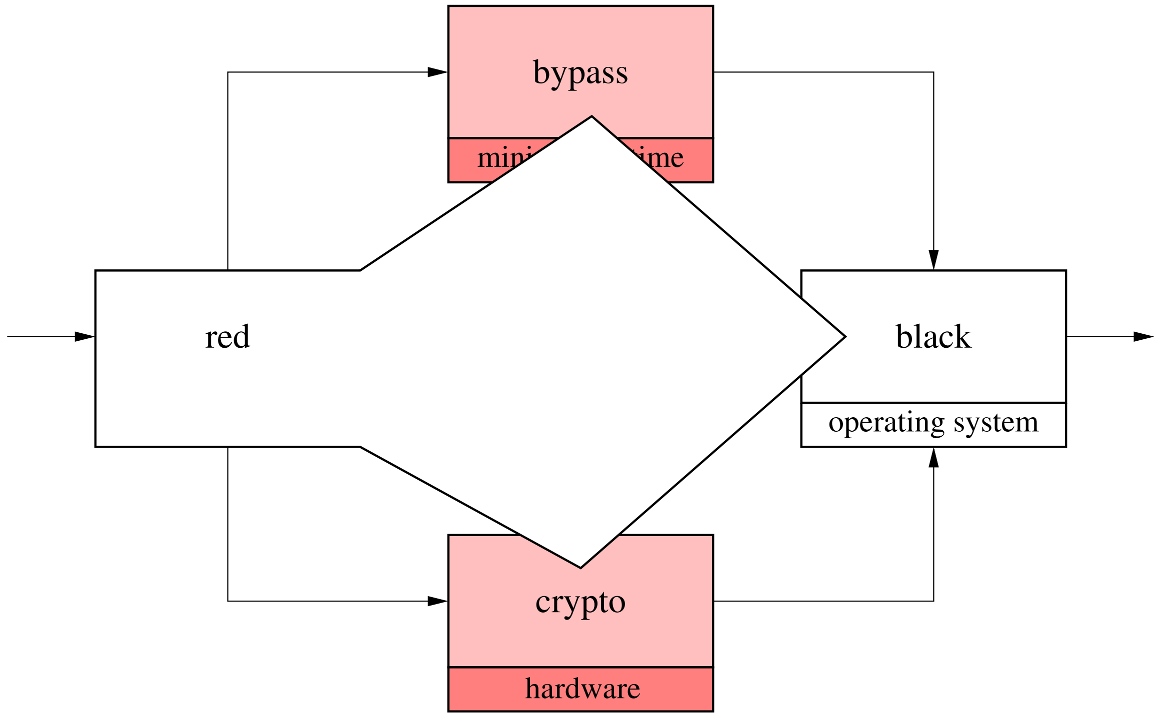


Naive sharing could allow direct red to black information flow, or could blur the integrity of the components

Unintended Communications Paths



Blurred Separation Between Components

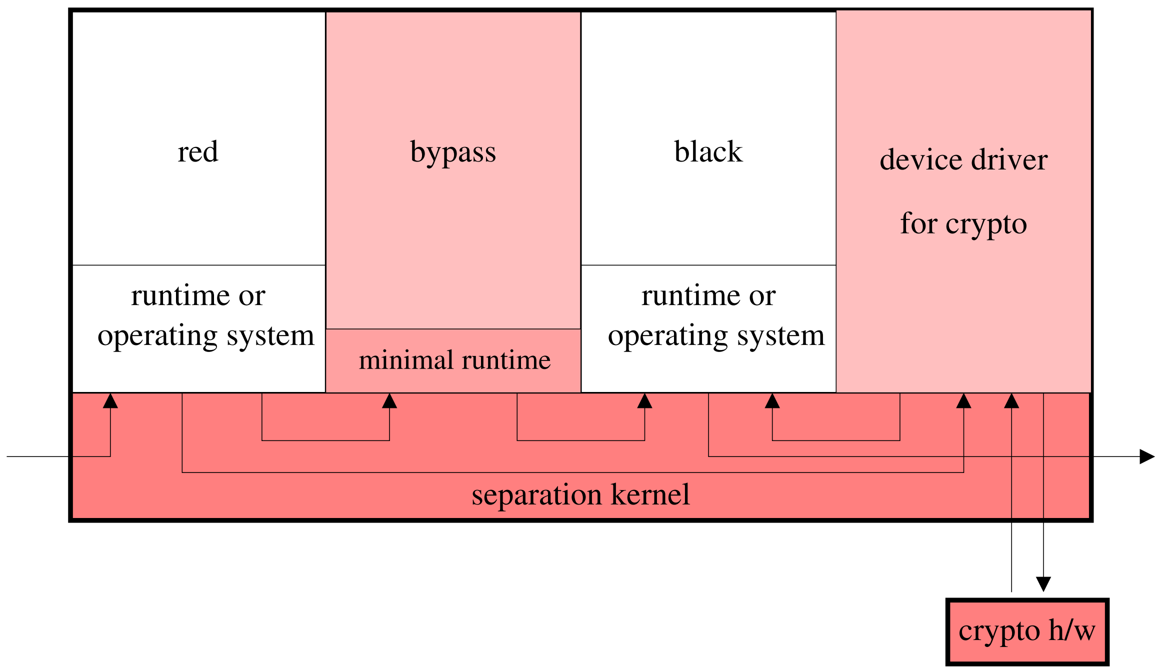


Secure Resource Sharing

- For broadly useful classes of resources
  - e.g., file systems, networks, consoles, processors
- Provide implementations that can be shared securely
- Start by defining what it means to partition specific kinds of resource into separate logical components
- Definition in the form of a protection profile (PP)
  - e.g., separation kernel protection profile (SKPP)
  - or network subsystem PP, filesystem PP, etc.
- Then build and evaluate to the appropriate PP

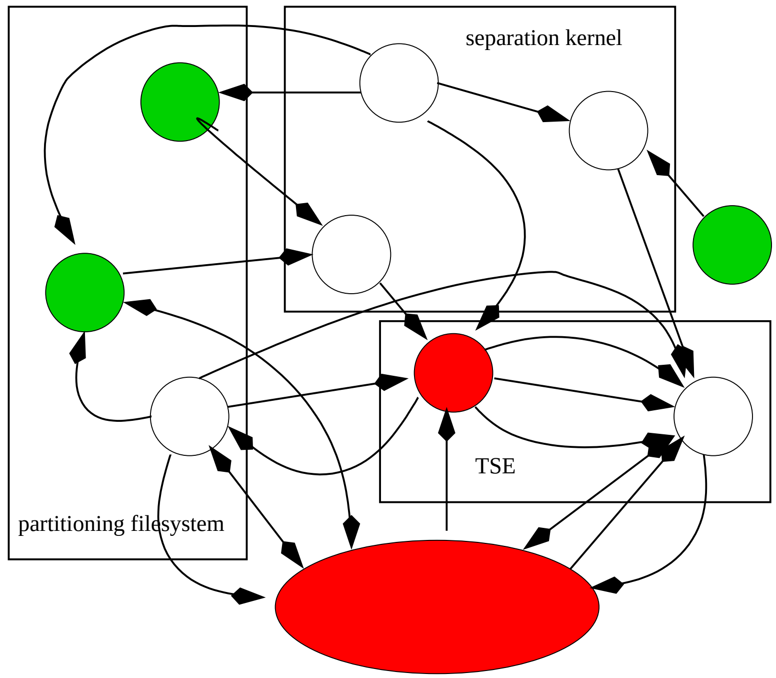
Crypto Controller Example: Step 3

Separation kernel securely partitions the processor resource



The integrity of the policy architecture is preserved

A Generic MILS System



Care and skill needed to determine which logical components share physical resources (performance, faults)

Resource Sharing: Compositional Assurance

- Construct assurance for each resource sharing component individually
  - i.e., each component enforces separation
- Then provide an argument that the individual components
  - Are additively compositional
  - e.g., **partitioning(kernel)** + **partitioning(network)** provides **partitioning(kernel + network)**

  And therefore combine to create the policy architecture
- Medium robustness: this is done informally
- High robustness: this is done formally
  - Compositional verification
- There is an asymmetry: partitioning network stacks and file systems and so on run as clients of the partitioning kernel
  - Hence, a link to the three-layer view

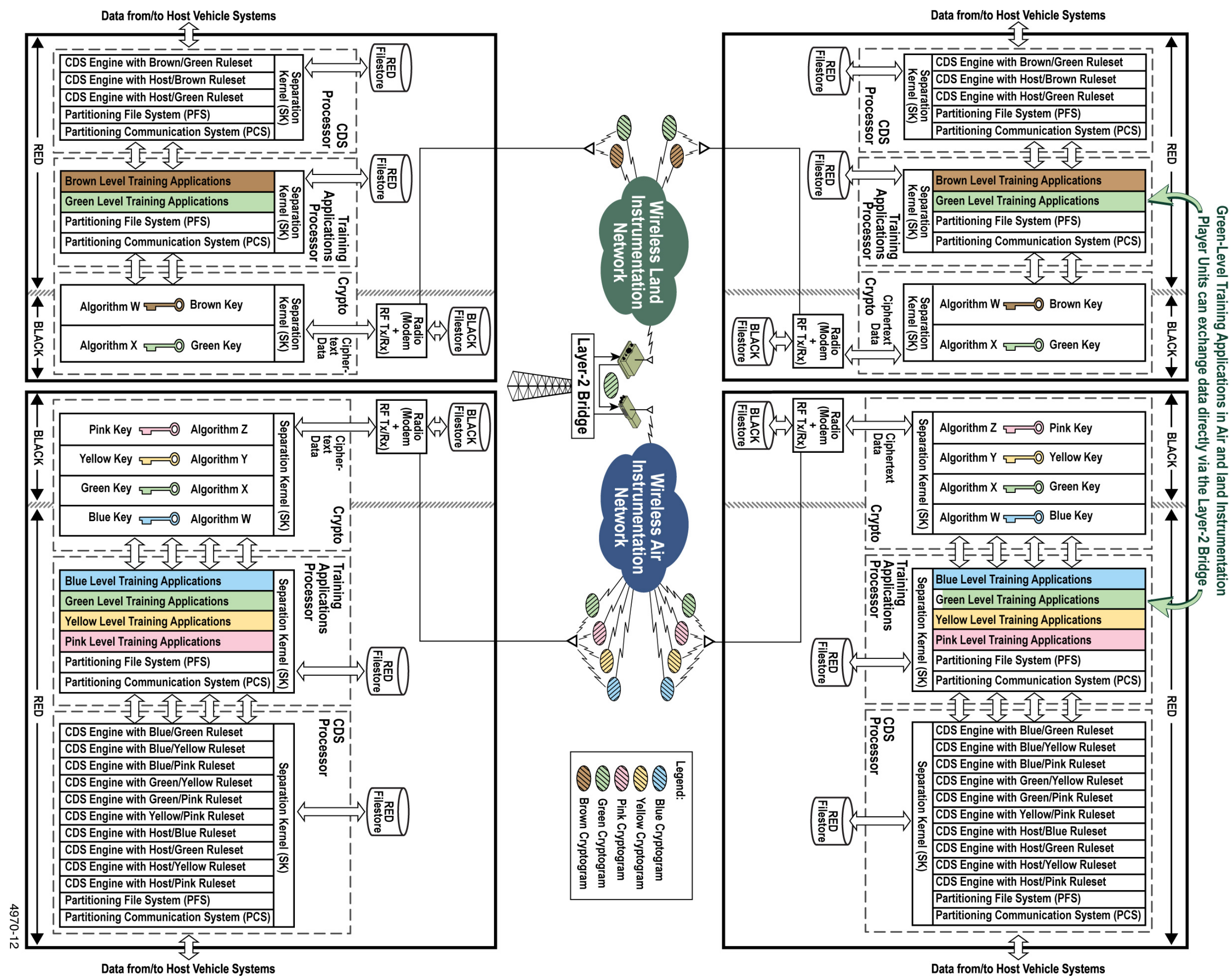
Compositional Verification: Resource Sharing Integration

- We have a formal policy architecture model
- Fairly standard Computer Science
  - Components are state machines
  - Communications channels are shared variables
  - Asynchronous composition
- Definition of well-formed policy architecture
- And of implementation respecting and enforcing a policy architecture
- Argument that these are additively compositional

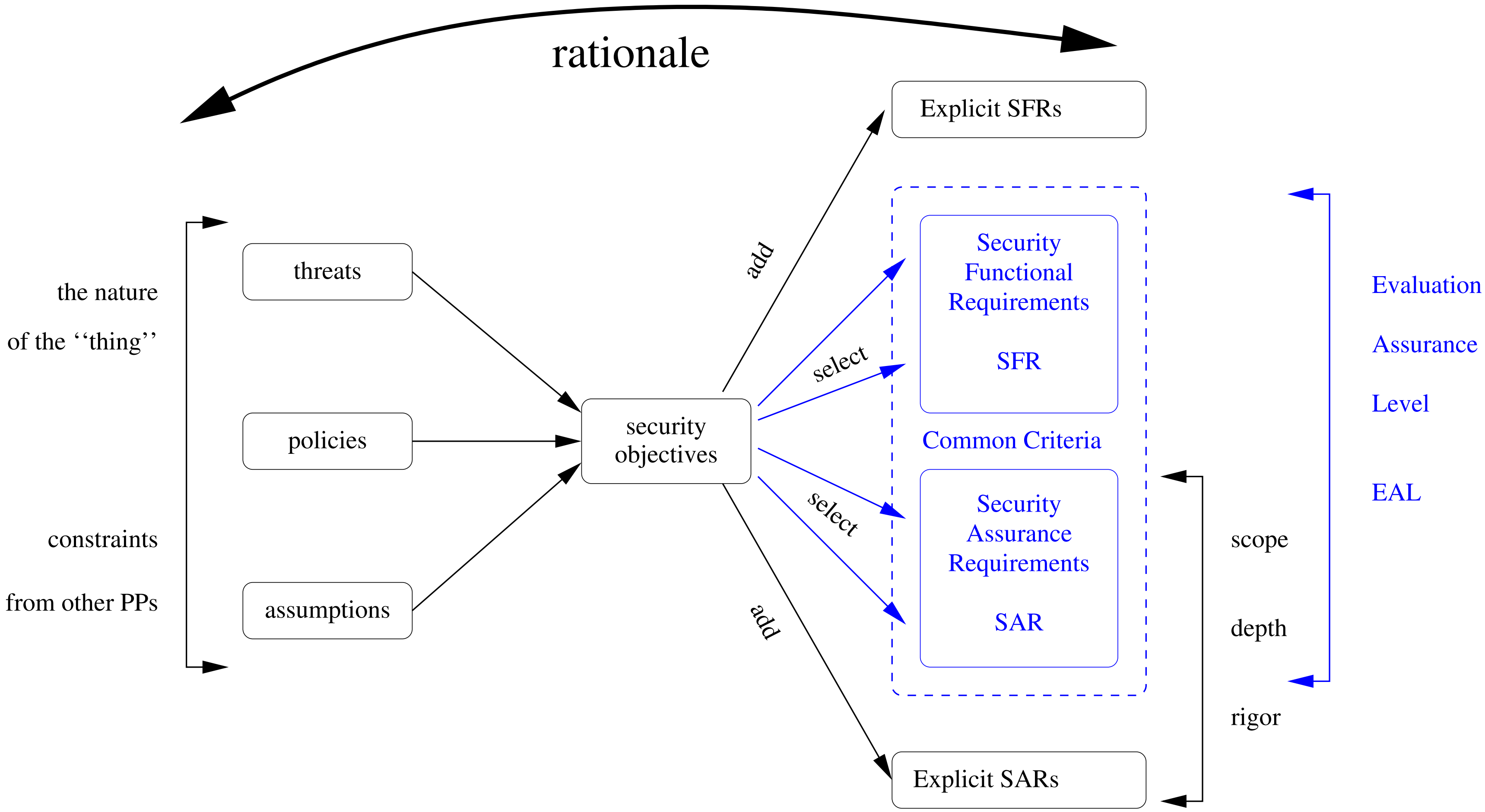
MILS Business Model

- DoD moves things forward by supporting development of protection profiles
  - Separation kernels, partitioning communications systems, TCP/IP network stacks, file systems, consoles, publish-subscribe
- Then vendors create a COTS marketplace of compliant components
- Currently they are all resource sharing components
- Should be some policy components, too
  - E.g., filters, downgraders for CDS
    - ⋆ Could be a standardized CDS engine, many rule sets
    - ⋆ Rule sets derived from goals, not hand coded
    - ⋆ e.g., Ontologically-driven purpose and anti-purpose
  - Or even MLS

Roadmap Example:
MILS Architecture for Joint Training Exercises



Protection Profile Development



A lot of delicate work

Protection Profiles Development (ctd.)

- Developing individual PPs is difficult and delicate work
  - Like developing a program against a big library
  - With no way to test it, except inspection
- Compositionality means PPs have to be collectively coherent
- We are developing a Common Criteria Authoring Environment (CCAE) to assist construction of coherent PPs
- Ontological characterization of SFRs and SARs and rules for their combination

MILS and IMA

- Certification for MILS is more demanding than for IMA
  - DO-178B Level A is comparable to EAL 4/5
  - High Robustness is EAL 6+/7+
- So a separation kernel is more aggressively minimized than a partitioning IMA RTOS
- But the basic ideas are very similar
- And the MILS approach to compositional assurance might apply to IMA integration

MILS In The Enterprise (e.g., Ground Systems)

- Separation kernels are like minimal hypervisors (cf. Xen)
  - MILS separation kernel (4 KSLOC), EAL7
  - Avionics partitioning kernel (20 KSLOC), DO-178B Level A ($\approx$ EAL4)
  - Hypervisor (60–250 KSLOC), EAL?
- Can expect some convergence of APIs (cf. ARINC 653)
- Different vendors will offer different functionality/assurance tradeoffs
- Could extend hypervisors from providing isolated virtual hosts to supporting the policy architecture of a secure service

Summary

- The MILS approach seems a reasonable approach to compositional reasoning about secure resource sharing and functional policies
- Must also consider resource utilization
  - Need tools to allocate/schedule resources such as processor time, bus access, IPC, devices
  - Given adequate specifications
  - These are fairly simple constraint satisfaction problems (e.g., CoBaSA)
- And fault propagation
- I think the approach can extend from security to safety
- And from embedded (airborne) to enterprise (ground) systems