MLS-PCA: A High Assurance Security Architecture for Future Avionics

Clark Weissman
Northrop Grumman Corporation
weissman@nrtc.northrop.com

Abstract[1]

DOD Joint Vision 2020 (JV2020) is the integrated multi-service planning document for conduct among coalition forces of future warfare. It requires the confluence of a number of key avionics technical developments: integrating the network-centric battlefield, management of hundred thousands of distributed processors, high assurance Multi Level Security (MLS) in the battlefield, and low cost high assurance engineering. This paper describes the results of a study and modeling of a new security architecture, (MLS-PCA), that yields a practical solution for JV2020 based upon DARPA Polymorphic Computing Architecture (PCA) advances, and a new distributed process-level encryption scheme. The paper defines a functional model and a verified formal specification of MLS-PCA, for high assurance, with the constraints PCA software and hardware morphware must support. Also, the paper shows a viable mapping of the MLS-PCA model to the PCA hardware. MLS-PCA is designed to support upwards of 500,000 CPUs predicted by Moore's law to be available circa 2020. To test such speculation, the paper concludes with a description of an in-progress proof-of-concept implementation of MLS-PCA using a 100-node Grid Computing system and an MLS distributed targeting application.

1. Introductions and Motivation

DOD Joint Vision 2020 describes the future battle space consisting of space, air, land, sea, and undersea forces integrated via a global network of sensors, command and control, communications, and integrated strike warfare elements [1]. The Achilles heel of this network-centric vision is the high assurance Multi Level Security (MLS) that permits the myriad communications that make JV2020 possible. MLS research and development over the past two decades has defined the requirements that must be satisfied for DOD systems [2, 3, 4, 5]. However, the high cost of developing and certifying high assurance systems to these requirements has been prohibitive and development time has been excessively long. Innovative use of Polymorphous Computing Architecture (PCA) to satisfy these MLS requirements in a scheme at process-level granularity is a novel R&D approach that simplifies system design, yet provides flexible configurable MLS systems. Such systems can meet security requirements to support different secure data streams in battlefield network-centric computing, as advocated in Joint Vision 2020. Many additional security requirements can be satisfied concurrently, including message integrity, authentication, confidentiality, code mobility, and dynamic coalitions. This paper describes a new security architecture to employ the richness of processing logic expected by 2020, such as the DARPA Polymorphic Computing Architecture (PCA) program [6].

The PCA program goals are to span a broad dynamic application space by implementing a transparent reactive layer between an embedded avionics application program and the malleable micro-architecture elements on which it will operate. This polymorphic layer will enable software and hardware to be developed in a cooperative constraint sensitive environment instead of in a failure prone hardware first and software last paradigm. The PCA program will implement a family of novel malleable micro-architecture processing elements, i.e., PCA chips, to include compute cores, caches, memory structures, data paths, network interfaces, network fabrics with incremental instructions, OS, and network protocols. These elements will have the ability to reconfigure to match changing mission and scenario demands. To support the use of polymorphous computing systems, the

[1] MLS-PCA is the acronym for "Multi Level Secure - Polymorphic Computer Architecture" the result of a study partially supported by DARPA through the Air Force Research Laboratories, USAF under agreement number F33615-01-C-1891, "Security/Trust as a Polymorphic Computing Constraint"

program will create a model based software framework for reactive monitoring, optimization, modeling, resource negotiation and allocation, regeneration, and verification.

Our new security architecture, MLS-PCA, is the focus of this paper, which covers the novel functional architecture, a formal specification model, and an examination of the security constraints that must be imposed on the PCA software-morphing layer and on the underlying chip hardware. The paper concludes with a description of a proof-of-concept demonstration using Grid Computing.

1.1 Character of Avionics

Legacy military avionics systems were developed using a "federated architecture" in which each subsystem was logically and physically separate. Each had its own set of component parts, which could not be used to support other subsystems in times of equipment failure. This was the approach taken with the F-15, F-16 and F/A-18, in the 1970s. In the early 1980s, the Department of Defense put together the "Pave Pillar" architecture that led to the Joint Integrated Avionics Working Group (JIAWG) Advanced Avionics Architecture. The result of this integrated avionics architecture was that computational resources could be interconnected by high speed networks to allow for more flexible usage of these resources, e.g., re-assigning a processor to take over the function of a failed processor. This also led to the ability to share information, e.g., to utilize fusion methods to merge radar and electro-optical information to create an improved way to convey information to the pilot. The pilot no longer had to mentally perform the integration function from a variety of gauges and instruments. Unfortunately, the sharing of information resources in a classified avionics environment leads to another challenge; either 1) operate at "System High", with a labor intensive burden of separating out the different classification levels at the end of a mission, or 2) solve the MLS problem. The combination of highly classified data along with un-cleared (or lowly cleared) maintainers led to a major Information Assurance nightmare. Methods currently do not exist to provide high assurance separation of the different security levels.

Future avionics systems will consist of a large number of processors interconnected by LANs, fiber channels, and local buses. Avionics application software – navigation, flight controls, communication, displays, targeting, and weapons control – will operate in a distributed manner, with processes spread across thousands of processors. Humans will play a variety of roles in this environment including pilot, navigator, ground controller, ground support, and mission planner. There is also a trend toward autonomous vehicles, where there is no authority to supervise security decisions. The growing need to use multilevel systems in coalition environments makes this a "show-stopper" issue!

1.2 System High Won't Work

Economics of general purpose computing has forced a tradition of developing software to share the processor resources. Operating systems, memory management, stack management, context switching, and interrupt vectoring are some examples of such sharing mechanisms. When avionics applications process different security levels of information, these sharing mechanisms must be trusted not to leak classified information between the processes. Trusted software is costly to develop, complex to design, poor performing, and difficult to certify its trustworthiness. As a result, most avionics systems avoid trusted development by operating at "System High," the highest classification of any data entering the system. In the future world of the integrated battlefield, System High is not an acceptable solution. Weapon systems, sensors, and people will create multiple secure data streams at different security sensitivities, which must be managed in a MLS manner to permit battlefield flexibility of application of those assets, and not over-classify information to System High. We simply cannot clear all battlefield personnel to System High. High assurance MLS is a necessary requirement because of the hostile battle space environment consisting of data as high as Top Secret with multiple compartments serving friendly forces and includes uncleared, foreign coalition partners, Red Cross, and humanitarian personnel; the worst case by current standards [7, 8].

2. MLS Problem

Simply stated, there are few Commercial Off The Shelf (COTS) solutions to satisfy the high assurance MLS requirements. The traditional alternative is to scratch build a high assurance trusted MLS system. That alternative is not attractive because 1) the avionics requirements are quite broad to meet all needs of the JV2020 battle space, 2) Certification and Accreditation (C&A) in DOD is in some disarray, with many competing approaches [9, 10, 11, 12, 13, 14], 3) obtaining C&A is a lengthy process that may not complete by time of need, 4) systems may not satisfy real-time avionics needs, and 5) traditional MLS approaches are too expensive. A new approach is needed.

In July 1990, the National Security Telecommunications and Information Systems Security Committee (NSTISSC) was established for the purpose of developing and promulgating national policies applicable to the security of national security telecommunications and information systems. In January 2000, NSTISSC issued Policy No. 11, which addresses the national policy governing the acquisition of information assurance and information assurance-enabled information technology products. Policy No.11 states that information assurance shall be considered as a requirement for all systems used to enter, process, store, display, or transmit national security information. DOD has issued DOD Directive 8500.1, Information Assurance, and DOD Instruction 8500.2, Information Assurance Implementation, to implement Policy No. 11 [15, 9, 10].

NSA and the Air Force have touted a trusted Protection Kernel (PK) as a candidate approach. They are supporting the development of a Common Criteria Protection Profile; a first step toward C&A [16]. PK divides a processor into isolated domains with controlled inter-domain communication. A different security-level process can run in each partition. There is prior research encouraging this approach [17]. COTS PKs available have weak security trust, and have never been applied to secure avionics application.

3. Moore's Law Predicts A Wealth of CPUs

A modern aircraft today has over 1000 computers on board, and many more on the ground in support of the vehicle mission. These are packaged into discrete systems with shared power systems, interconnect busses, and external communications. The computers perform flight control management, navigation, stores management, sensor processing, targeting, weapons control, communications processing, and display processing. Collectively, the high-level language software for these functions is multiple millions of source lines of code (SLOC), and rising with new developments.

For the past 30 years, computer hardware logic per chip has been growing exponentially, doubling every 18 months. First formulated as Moore's Law, the forecast in 2002 is for the exponential growth to continue with 12 logic doublings by 2020 (2^{12} = 4,000 X), our target timeframe [18]. The security challenge will be to build a high assurance MLS system from the expected array of 400,000 processors. We will be in an era of logic – processor/memory – richness. This paper proposes one way to securely organize and employ this computational richness.

4. MLS-PCA Characteristics

Key to avionics security is creating dynamic trusted connections <u>between processes</u>, not between processors, as was achieved in the Defense Data Net (DDN) [19], and is typical with today's network Virtual Private Network (VPN) architecture. Cryptography is placed at the junction of processors – host, router, server, firewall, and gateway – or within the processor with unknown quality encryption software, or as software mediated cryptographic chips. Alas the rub, all these approaches place the security base on untrusted software intermediaries.

Future computing will have processor- and memory-rich avionics designed as distributed processes in a plexus of processors interconnected by networks. Our approach is to move encryption to the process level to create trusted application connections with unique trusted cryptographic elements. The operative components of the architecture are an Encryption Process Element (EPE) interposed between an Avionics Application Process (AAP) and the communication channel. A Network Security Element (NSE) will control the Inter Process Communication (IPC) via distribution of encryption and authentication keys to the EPEs.

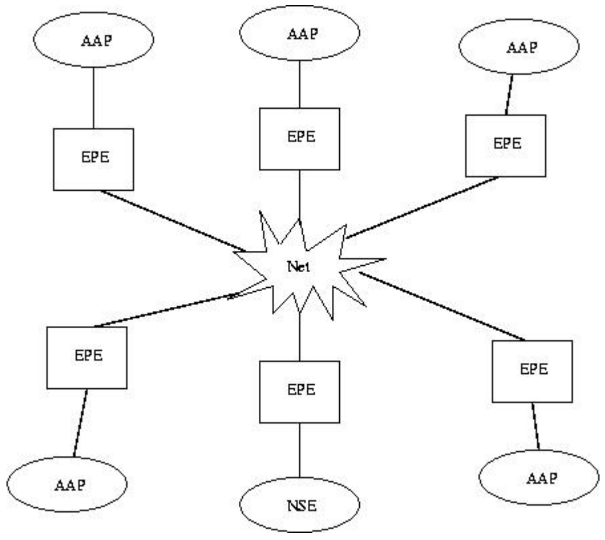


Figure 4. MLS-PCA Example Network

An AAP is hosted on its own processor. There is no need to share the processor and its resources with another AAP. There is no need for a complex resource manager or Operating System. A simple network protocol stack and loader is sufficient. Domains and domain management, e.g., context saving, context switching, are unnecessary. Memory management and sharing are eliminated as well as process scheduling. The absence of these features permits simpler hardware and CPU architectures, perfect for high density, multiple processor chips now coming online, i.e., DARPA PCA. And the greatest benefit for security is the dedication of each processor to a single security level, that of its loaded AAP process.

A small example of a MLS-PCA secure network is shown in Figure 4. Note the pairing of each AAP-EPE, including the NSE-EPE pair in bottom center of figure.

5. MLS-PCA Functional Model

Avionics components are usually well defined by the mission and include air vehicle controls, navigation, (e.g., Global Positioning Satellite, GPS), inertial, targeting, sensor (e.g., Infra Red, IR, radar), weapons control, payload stores, communications, safety, and other systems. Ground support functions include maintenance and logistics, mission planning, mission analysis, and training among others. These support functions affect the avionics configuration. Mission planning determines flight plan, weapons, radio frequencies, crypto keys, weather, targets, etc. Plans so formulated are embodied in software programs and databases that are dynamically loaded into the air vehicle just before takeoff by some Portable Memory Device (PMD) carried by the pilot or crew.

5.1 Avionics Application Process, AAP

The avionics development includes infrastructure components – processors, busses, communications devices, etc. – under control of the appropriate application software processes. We define these as Avionics Application Processes, AAPs. Traditionally, AAPs are integrated into one large system operating at the system high classification of the vehicle, e.g., Top Secret Special Access Required (TS-SAR). MLS-PCA will require different thinking on the part of avionics developers. Functions will be classified individually at the single level of the highest data processed, often less than system high. Thus AAPs are the untrusted "subjects" of the Bell-LaPadula access control policy model [20], and will be at a variety of security levels, mostly Unclassified or Secret. Mission planning will select the required software for the mission, and construct a table – the access matrix – of the AAPs, which will specify their security levels, the data and devices, i.e., the "objects," they can access, and the type of access permitted, i.e., their read, write, append, and execute permissions. Furthermore, mission planning will define the avionics system configuration of network addresses, process ids, authenticators, and initial cryptographic keys. This classified data is protected from theft, unauthorized modification, and disclosure by encrypting the PMD for its journey from the classified mission-planning center to the classified air vehicle and back again after the mission with mission results.

An AAP is considered a homogenous process at a single security level. In reality, it may be many processes, but packaged for MLS-PCA as a single process. For real-time systems, an AAP traditionally is scheduled to run at a precise time interval by an event trigger, or by a call from another AAP. For MLS-PCA, the AAP will own its processor exclusively and need not be scheduled. It will always run, but only produce results when events dictate. When necessary, an AAP will interact with another authorized (by the access matrix) AAP. MLS-PCA will establish a cryptographically "trusted connection" between the two AAPs. Multiple AAPs can share a trusted connection as part of a "coalition." AAP trusted connections could last the entire mission, and often will in the well-defined world of avionics. Finally, the trusted connection can extend beyond the boundary of the avionics "box," or the air vehicle when properly configured. The trusted connection is only limited by the communications and imagination of the system developer.

5.2 Encryption Processing Element, EPE

Each AAP will be protected by an "attached" front-end guard element, the EPE. The EPE guards the attached process by performing message encryption/decryption of all IPC traffic. There is no bypass of the EPE. This is a security constraint on the architecture, the guarantee that a cryptographic computing element front ends each computational element. An EPE may be a software element or encryption hardware. There can be thousands of EPEs at any given time. An EPE does additional tasks related to protecting keys as a way of enforcing security policy. For example, all keys are distributed "wrapped," i.e., encrypted. The EPE must unwrap keys to use them. The wrapper key must be distributed in an "out of band" procedure, possibly carried in a physical "ignition" key generated by mission planning, and inserted into an avionics port by the pilot, or built into each EPE processor's nonvolatile memory by mission control. The choice is mission dictated and hardware configured.

In summary, each AAP has one EPE. The EPE is the only access between the AAP and the communications network and functions as a gateway to ensure that messages can be sent only to authorized recipients and that all messages are encrypted.

5.3 Network Security Element, NSE

The NSE distributes encryption keys to the EPEs, enforcing access control of communication paths, i.e., permissions between AAP pairs. The NSE is the security policy element for all internal and external communication, permitting the avionics interoperability with external battlefield assets. Within the control of the NSE is an access matrix of authorized permissions for each AAP. The permissions are stored as a database, with a unique key corresponding to each dimension of the security policy. For example, there can be a key for each security level and each compartment of the Mandatory Access Control (MAC) security lattice. There can be a common key for each user (uid) or process (pid) in a coalition, or a key for each AAP pair allowed to connect as part of Discretionary Access Control (DAC). There can be keys for each mission function, and there can be one-time session keys for each newly created trusted connection. NSE creates a trusted connection, by sending a session key to the attached EPEs. That session key is the XORed result of all the policy keys – the MAC, DAC, and other keys – for the connection based on the maximum authorized permission of the paired application processes. The NSE access matrix is authorized and established by mission planning and transported to the avionics system on a PMD at mission initiation. Dynamic updates are permitted by authorized roles in the mission, e.g., pilot, and/or ground control.

At mission initialization, system required trusted connections are established between security infrastructure elements – NSE, EPE (cf. Section 5.6.2). They exist to allow the NSE to distribute keys securely to EPEs. Information regarding AAPs is required for setting the NSE access database at mission initialization. A human role is defined by associating a user (uid) with a process (pid) in the access control matrix. For each pid and uid there is a set of credentials that defines the security permissions, the coalitions, and the roles played by all entities. There is a need for Identification and Authorization (I&A) whenever connections are established. The NSE will perform the I&A task inasmuch as it already has the I&A data from mission planning. The NSE can be implemented as a set of distributed processes executing on multiple processors within the avionics architecture for redundancy and performance, similar to any of the avionics applications.

5.4 Security Policy Enforced by Encryption

The enforcement mechanism of the MLS-PCA model is the allocation of an encryption key for the trusted connection between two AAPs – the session key, $K_{session}$. The NSE computes the session key for each open request by an AAP to access another, based on the applicable security policy. Typically, there are multiple applicable policies – MAC, DAC, and Mission.

MLS-PCA treats AAPs as untrusted subjects, and treats trusted connections (TCs) as the security objects. TCs are simplex (unidirectional), i.e., AAP_i can write messages to AAP_j (who reads messages from the connection). If AAP_j wishes to respond to AAP_i, AAP_j must open a separate simplex connection to AAP_i. Most dialogs between AAPs will be "duplex" by creating two simplex connections. Simplex connections allow blind write-up, or Append, e.g., AAP_i may write to AAP_j, when the security level $SL_j >= SL_i$ (dominates).

Mandatory Access Control, MAC, is the classic DOD policy of a subject's clearance dominating an object's classification. This is best realized in the Bell-LaPadula [20] policy. MLS-PCA uses Bell-LaPadula and labels all subjects and objects. There is a MAC key, for each classification level, K_{sl}, and each security compartment, K_{comp} .

Discretionary Access Control, DAC, further limits subject-object access. DAC is like a "wiring diagram" of mission functions (AAPs). DAC is conceptualized as a matrix of subjects vs. objects, with a matrix cell's content containing the DAC encryption key, K_d. The DAC matrix is sparsely populated because the AAPs tend to cluster by function. For avionics purposes, a coalition is a collection of subjects who meet the requisite MAC requirements and are members of a community of interest of the MLS-PCA model. These subjects create a multi-party trusted connection by joining a coalition and leaving the coalition as necessary. MLS-PCA effects a coalition by treating coalitions as objects in the DAC matrix and creating a common key K_{coal} used by all coalition subjects. For each subject in a coalition, its coalition key, K_{coal}, is contained in the DAC matrix coalition cell. Thus, DAC policy key K_{dac} is defined as $K_{dac} = (K_d \text{ or } K_{coal})$, i.e., either the DAC key or the coalition key for a given object.

The MLS-PCA model is applicable to a wide family of avionics applications in a dynamic battle space environment. Missions can cover surveillance, targeting, shooter, and communications. MLS-PCA takes the view that an avionics mission is composed of a set of AAPs that constitute the mission functionality. The mission can then be represented by the DAC policy above. For multi-mission scenarios we need another (3rd) dimension to the DAC matrix that shows the DAC connectivity for each mission, i.e., another layer in the DAC matrix.

Overall then, the MLS-PCA security policy is reflected in the following:

$K_{session} = K_{sl} \otimes K_{comp} \otimes K_{dac}$[2];
where, $K_{dac} = (K_d \text{ or } K_{coal}) \otimes K_{mission}$
and $\otimes$ is XOR

This scheme provides great flexibility in MLS-PCA to match security policy to the needs of the avionics application. Most missions are static with fixed AAP communication patterns as one might find in an autonomous Unpopulated Air Vehicle (UAV). In such a static environment, we might do away with the NSE and have access policy keys pre-placed during initialization at the EPEs by mission control.

5.5 Crypto Issues

The MLS-PCA model is silent on how the encryption function is mechanized – in software or hardware. It is only concerned that it be correct, always invoked, and always bound to its AAP. It is the "reference monitor" for the architecture [2].

The model is also silent on the encryption algorithm to be employed. We only assume it will have management features compatible with DOD Type I and Type II encryption, and commercial algorithms such as Triple Data Encryption Standard (DES), and the Advanced Encryption Standard (AES). Choice will be made at the time of specific application. We do specify a Public Key Infrastructure (PKI) scheme for secure key distribution during system boot (cf. Section 5.6). Key management is intimately tied to security policy as discussed in Section 5.4.

Every secure system must have a means of revoking access upon discovering hostile, or runaway behavior of a subject. This means revoking a trusted connection immediately. Revocation is achieved by erasing the guilty AAP connection by "zeroizing" the session key for the connection, $K_{session}$.[3] Zeroizing is a command sent from the NSE to the EPE guarding the guilty AAP. Since the NSE and EPE are trusted processes the key erase action occurs near instantaneously breaking the AAP trusted connection. The AAP cannot thwart the zeroize action because it is not a party to the private infrastructure command between the NSE and EPE. Also, unlike zeroize of traditional encryption boxes, the zeroize command can be acknowledged and states synchronized after action taken by the EPE, which has a separate trusted connection with the NSE. The model also uses zeroize of K_{coal} at a specific EPE to remove a subject (i.e., AAP) from a coalition.

5.6 Initialize and Bootstrap of MLS-PCA

The NSE and the EPE is the Trusted Computing Base (TCB) for the MLS-PCA scheme. There are two possible implementation configurations for the MLS-PCA model to protect the TCB: the first has the EPE in hardware; second, has the EPE as a loadable software process. Our view of the first consideration is the EPE process is a hardware subroutine of the CPU chip, somewhat like floating point hardware. We are looking at the proposed PCA hardware chips for MIT's Raw [21] and Stanford's Smart Memories [22] for how the model maps into the hardware. Generally speaking the hardware configuration is an easier initialization implementation because most of the initial parameters are "wired" into the hardware, e.g., network addresses, or process logic. The unique hardware initialization issues are a resource allocation consideration when there exist lots of CPUs, memory, and buses on a chip, i.e., a Raw chip has 16 CPUs; Smart Memories has 64 CPUs. The software EPE initialization issues are classical security and integrity issues, the harder solution of the two configurations.

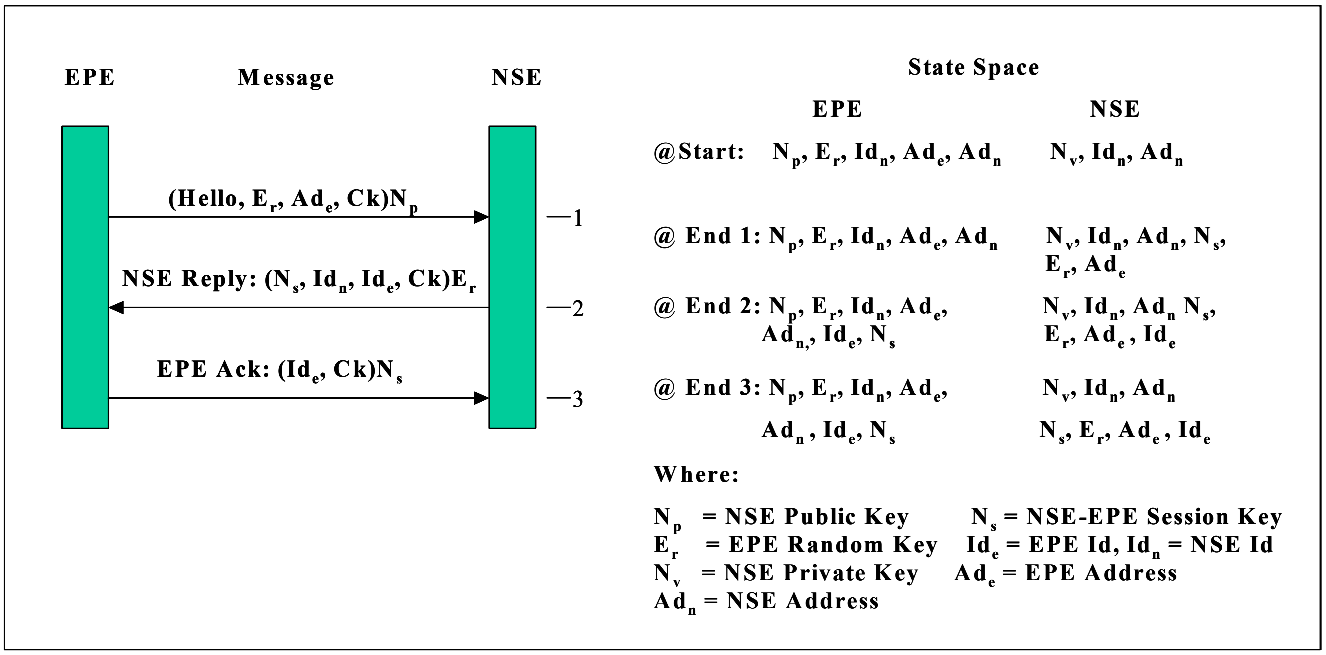
5.6.1 Assumptions. For any given classified avionics environment, the classified data and applications (AAPs) will be created and configured in a classified and trusted ground-based support system, a Mission Planning Center (MPC). The MPC is an MLS trusted facility that plans the mission, assembles the avionics mission software AAPs from trusted configuration files, and defines the mission configuration parameters (i.e., AAPs, NSE, EPEs, flight plan, radio frequencies, encryption keys, security levels of AAPs, weapons and fuel stores, and other items). The mission vehicle information systems will contain only unclassified data when "parked," be it an aircraft, UAV, or ship. The mission configuration parameters will be written to a PMD to be loaded into the vehicle just prior to the mission. The PMD will be encrypted to

[2] Added security can be achieved by applying a non-invertible function to $K_{session}$ to foil a rogue process impersonating an EPE from obtaining $K_{session}$ and deducing the component keys.

[3] "Zeroize" does not mean setting a key of all zeros. It means replacing a key with a random value not known by any other EPE, thereby making encrypted text using the zeroized key undecipherable.

protect the pre- and post-mission information stored on the PMD.

There is a well known problem in trusted systems we call the “fixed point theorem.” Encryption keys can be wrapped in other encryption keys for protection during transmission and storage outside of the crypto component. However, at some fixed point there is a secret clear text key pre-placed to permit the boot process to unfold in a staged and protected manner. In MLS-PCA the fixed point is a physical “ignition key” inserted into the system, and a pre-placed PKI private key in non-volatile memory of the NSE processor board, similar to the Trusted Computing Platform Alliance scheme [23]. The ignition key is used to begin the unwrapping of encrypted keys using a physically protected token. To decrypt the PMD, the ignition key will be carried to the vehicle by the pilot (or mission commander for pilot less vehicles) and inserted in the cockpit prior to takeoff. The ignition key, like the PMD, is created by the MPC. We anticipate NSA as responsible for the PMD encryption/decryption logic and wiring of the ignition key reader and PMD.


EPE
Message
NSE
$(\text{Hello}, E_r, Ad_e, Ck)N_p$
NSE Reply: $(N_s, Id_n, Id_e, Ck)E_r$
EPE Ack: $(Id_e, Ck)N_s$
—1
—2
—3

State Space

	EPE	NSE
@Start:	$N_p, E_r, Id_n, Ad_e, Ad_n$	N_v, Id_n, Ad_n
@ End 1:	$N_p, E_r, Id_n, Ad_e, Ad_n$	$N_v, Id_n, Ad_n, N_s, E_r, Ad_e$
@ End 2:	$N_p, E_r, Id_n, Ad_e, Ad_n, Id_e, N_s$	$N_v, Id_n, Ad_n\ N_s, E_r, Ad_e, Id_e$
@ End 3:	$N_p, E_r, Id_n, Ad_e, Ad_n, Id_e, N_s$	$N_v, Id_n, Ad_n\ N_s, E_r, Ad_e, Id_e$

Where:

N_p = **NSE Public Key** N_s = **NSE-EPE Session Key**
E_r = **EPE Random Key** Id_e = **EPE Id,** Id_n = **NSE Id**
N_v = **NSE Private Key** Ad_e = **EPE Address**
Ad_n = **NSE Address**

Figure 5.6: EPE Initialization Protocol

Typically, there will be one NSE and thousands of AAP-EPE pairs. The NSE may be redundant or distributed for reliability. The boot logic for the system will have the NSE loaded first, followed by prioritized EPE-AAP pairs loaded from the PMD. The mission will drive all the initialization parameters. MPC will determine the load priorities, locations of all devices and processes (i.e., their net addresses, Ad_n and Ad_e), their identifications (Id_n, Id_e), and the PKI private key and public key of the NSE (N_v and N_p, respectively). MPC will also build a table of permissions and classifications for all AAPs, the Bell-LaPadula access matrix, for the mission. Lastly, the NSE will know all these initial conditions by loading the access matrix from the PMD; the EPEs will know some of these data by parameter loading by MPC or NSE for each EPE-AAP pair code loaded – N_p, Ad_n, Ad_e, Id_n.

5.6.2 EPE-NSE Initialization Protocol. There can be a priority of operation of the mission functions reflected in the order of AAP initialization. The NSE will know that priority. For an AAP to run it must first be bound to an EPE. Since both AAP and EPE are software processes, they should run on adjacent processors. Also, there is nothing unique about an EPE; any EPE can be bound to a unique AAP. The NSE reads the PMD and creates an EPE, loading and/or assigning parameters to bind it to an AAP. The EPE is then executed while the NSE creates another EPE. The EPE’s first action is to generate a random key (E_r). Since all EPEs are identical, E_r must be based on some changing system variable to avoid repeating E_r among different EPE invocations. Its next action is to create and send a Hello message to the NSE, giving the Hello message identification, its net address (Ad_e), random key (E_r), and an integrity checksum, all wrapped in the public key (N_p) of the NSE. This foils unauthorized reading of the Hello message by possible Trojans hidden in the architecture. The NSE saves these parameters and assigns the next priority AAP to this EPE by assigning an identity (Id_e) to the EPE; Id_e can be the identity of the bound AAP. It includes its own Id_n to confirm to the EPE its identity, gives a newly created NSE-EPE session key (N_s) based on the security level of the bound AAP, adds an integrity checksum, and wraps the whole message in the EPE's E_r. This provides critical information securely to the EPE to whom it is bound, including the session key for further NSE dialogs, the identity confirmation of the NSE, and an indication that a false NSE is not spoofing it. The last message by the EPE is an acknowledgement to synchronize state with the NSE. This complete initialization sequence and state space is shown graphically in Figure 5.6.

6. Certification and Accreditation, C&A

With the demise of DOD 5200.28-STD [3] and the NIAP practice defined only through Common Criteria (CC) Evaluation Assurance Level 4 (EAL4), high assurance (EAL5-7) is without a C&A support organization. Only a DOD user agency can assume responsibility, and only for its application [11, 15]. Because MLS-PCA is years away from its first application, we have adopted the CC EAL7 C&A as the driving security assurance requirements, particularly the formal specification and verification [4].

Table 6.1: MLS-PCA Formal Spec Characteristics

Feature	Quantity
Signatures (i.e., Domains)	63
Relations (i.e., State Variables)	64
Operations (i.e., Transforms)	39
Predicates (i.e., Conditionals)	38
Facts (i.e., Definitions)	28
Invariants (i.e., Constraints)	18

6.1 Formal Specification

The functional model for MLS-PCA is described in Section 5. Early in the DARPA PCA program we studied available formal language systems [24] and selected MIT's Alloy for its state machine expressiveness, available tool suite, and its constraint checking approach to spec verification [25]. The MLS-PCA formal spec is now complete and verified. Its salient features are summarized in Table 6.1. Details of the formal specification can be found in the companion paper, "Using Alloy to Formally Specify MLS-PCA Trusted Security Architecture" [26].

6.2 MLS-PCA Verification

The Alloy Constraint Checker does not prove a spec is correct, rather it assures the user the spec is consistent with its assertions, constraints and initial values. It logically checks the spec to show the existence of values of state variables that satisfy all spec transforms, constraints, and conditions. If there are no values of state variables that meet these conditions, the spec is "over constrained." If there are contradictions among the states, constraints, and values, the spec is "under constrained." Finding the right balance between these extremes is the art of writing a formal spec. The Alloy Constraint Checker provides the engineering balance to verification between no checking and formal proof. The tools are fast and quite useable [27].

6.3 MLS-PCA Flaws Found by Verification

One of the earliest flaws found by the Alloy Constraint Checker we subsequently called "fate sharing" – death of an EPE and its bound AAP. Before repair, the spec allowed an AAP process to die and be replaced by another. The Alloy Checker found a case where a message destined for an old Secret AAP arrived and was delivered to a new unclassified AAP now bound to the EPE; a clear violation of the no write-down constraint. The repair was fate sharing of AAP-EPE.

Another flaw found dealt with messages arriving out of order. An NSE sends two messages to an EPE: rekey then revoke a trusted connection. If the messages arrive out of order, the new key would in effect re-establish the just revoked connection. MLS-PCA will guarantee messages arrive in order to assure state synchrony and prevent replay attacks.

The last example arises from the distributed nature of the MLS-PCA model. Distributed systems have potential state synchrony problems due to message delays. The Alloy Constraint Analyzer found a case where the operation that changes DAC permissions invalidated the DAC invariant, that says "a communication path is in existence only if the DAC policy allows it." The problem was the delay in synchronizing the DAC change with the EPE state – an example of the classical problem of when does a revocation become effective? Normally this invariant check is used to determine if an operation is under-constrained. Here, it is the invariant that is over-constrained. The solution was to relax the DAC invariant and apply it only to new connections. The new rule is " A connection path may be created only if the DAC policy allows it."

7. Conclusion: Proof of Concept

The proof is in the pudding, and our pudding is an MLS-PCA proof of concept demonstration. But how do you demonstrate thousands of networked processors nearly two decades before they exist? We wish to write code from the formal specs for the NSE, EPE, and a vital application of AAPs. Simulation would not admit code, and lashing together many microprocessors is too expensive. Using hundreds of PCA chips is years in the future when they first become available from DARPA. That was our dilemma early on. We found a way using Grid Computing [28].

7.1 Grid Computing: Simulating 1000s of CPUs

The DARPA contract under which MLS-PCA was developed was limited just to the formal modeling effort. However, Northrop Grumman Corporation found the modeling results of great interest and chose to sponsor a prototype using CY 2003 Independent R&D (IRAD).

The Northrop Grumman Corporation R&D private network consists of hundreds of user workstations of Windows PCs, Sun Solaris machines, and Silicon Graphics workstations. If we ran just 10 AAPs per workstation, we can have a 1000 node distributed MLS-PCA implementation via a Grid Computing architecture. The net is essentially available two shifts a day plus weekends. We wrote the NSE and EPE code in C++ from the formal specs. We run the code under Windows with an IP protocol stack, and build trusted connections with IPSec, 256-bit key AES software encryption, and HMAC-SHA-256 authentication for the EPE. We recognize the vulnerability of MLS-PCA to attacks on the Windows OS; however, the objective of this implementation is to shake out the model, demonstrate its soundness, and collect basic performance data. Later, we will drop Windows and replace it with a high assurance TCB to boot and run NSE, EPE and AAP code. We have also selected a meaningful MLS-PCA demonstration application.

7.2 Targeting Application

MLS-PCA computing demands a different paradigm for designing applications. Rather than construct large monolithic functional modules of hardware and software, as is current practice, our model demands functions be composed of small code segments, i.e., processes, each of which operates at a single security level within its own processor, and communicates with related processes via Inter Process Communication. The art of designing such distributed software is just beginning to take shape in various research efforts, e.g., DARPA's "agent-based systems"[29]. Northrop Grumman Corporation has an extensive R&D effort in multi-sensor target detection. These algorithm-based applications have MLS properties and can profit from the parallelism inherent in the distributed MLS-PCA model and Grid Computing.

A typical target radar or infrared image can be as large as 9,000 x 9,000 pixels. This is too large for processing algorithms on anything short of a high performance server or special digital signal processor; certainly beyond the capabilities of a current PC. Our Grid Computing demo will divide the image into 100 smaller 1,000 x 1,000 pixel sub-images or cells, allowing for cell overlap to avoid missing features that span a cell boundary, and pass each cell on to a Grid processor. A Grid processor is capable of applying a search algorithm to detect potential targets in the cell. We are simulating data for different target types. When a potential target is detected and identified in the cell, the algorithm will report the potential target to a central controller that will eliminate duplicate reports. The central controller will report each potential target to one of four target controllers, one for each of four types, also running as separate processors on the Grid. Target controller types are at different simulated security levels: unclassified, confidential, secret, and top secret. The cell processors will all run at a simulated unclassified security level. A cell processor will write up to the central controller's higher security level, and the central controller will write up to the appropriate target controllers. MLS-PCA is designed explicitly via the simplex trusted paths to permit authorized Append connections. The target controllers will display the reports from all the cells, and show the distribution of found targets on a composite system high display.

7.3 Performance Goals

The objective of the demonstration is to show the feasibility of building MLS systems on the MLS-PCA architecture. It will provide us with a vehicle for gathering performance data on the critical choke points in the architecture, the boot procedure, the initialization mechanisms, and oversights in the design. The demonstration will achieve:

- Simulated MLS operation in a distributed network

- Distributed application of a typical avionics function, i.e., targeting
- High value of formal specification
- Proof of concept for MLS-PCA
    - Operation of live NSE, EPE, and AAP code and IPC
    - Boot of MLS-PCA
    - Performance data on MLS-PCA operation
- Program interest from a real avionics application

There was insufficient testing to report our findings by publication date. However, the coding and checkout of the NSE and EPE went well and quickly, completed in four-months. The NSE is 3371 Source Lines of Code (SLOC) compiling to 344K binary, and the EPE is 2679 SLOC compiling to 580K binary, not counting libraries used. The non-optimized code is small enough to be handled by any of the Grid PCs, and even the limited PCA memory available with first generation chips. The applications AAPs are the long lead-time elements. A future companion paper will report the specific findings of the Proof of Concept Demonstration.

8. References

[1] "Joint Vision 2020", *JCS, J5,* June 2000. Available at http://www.dtic.mil/jointvision/jvpub2.htm.

[2] J. P. Anderson, "Computer Security Technology Planning Study", 1972, In *ESD-TR-73-51.*

[3] "Department of Defense Trusted Computer System Evaluation Criteria (TCSEC)", *DOD 5200.28-STD,* December 1985. Available at http://www.fas.org/irp/nsa/rainbow.htm

[4] "Common Criteria for Information Technology Security Evaluation", ISO*/IEC 15408, Version 2.1, CCIMB-99-031*, August 1999. Available at http://www.radium.ncsc.mil/tpep/library/ccitse/ccitse.html.

[5] Rainbow Series of books on evaluating Trusted Computer Systems according to National Security Agency (NSA) expounding on the Orange Book (TCSEC). Available at http://www.fas.org/irp/nsa/rainbow.htm

[6] "Polymorphic Computing Architecture Mission" Available at http://www.darpa.mil/ipto/research/pca/

[7] "Computer Security Requirements – Guidance for Applying the DOD TCSEC in Specific Environments", June 1985*, CSC-STD-003-85.* Available at http://www.fas.org/irp/nsa/rainbow.htm

[8] "Technical Rationale Behind CSC-STD-003-85: Computer Security Requirements", June 1985, *CSC-STD-004-85*. Available at http://www.fas.org/irp/nsa/rainbow.htm

[9] "Information Assurance", October 2002*, DOD Directive 8500.1.* Available at http://www.dtic.mil/whs/directives

[10] "Information Assurance (IA) Implementation", February 2003, *DOD Instruction 8500.2,* Available at http://www.dtic.mil/whs/directives

[11] "National Industrial Security Program Operating Manual, NISPOM", *DOD 5220.22-M,* December 1993. Available at http://www.dss.mil/infoas/index.htm

[12] "Protecting Sensitive Compartmented Information within Information Systems", *Director of Central Intelligence Directive 6/3*, June 1999**.** Available at http://www.fas.org/irp/offdocs/DCID_6-3_20Policy.htm

[13] "DOD Information Technology Security Certification and Accreditation Process, DITSCAP", December 1997, *DOD 5200.40,*. Available at http://www.dss.mil/infoas/index.htm

[14] "National Information Assurance Partnership, NIAP", *NIST,* 1997. Available at http://niap.nist.gov/

[15] "National Security Telecommunications and Information Systems Security Committee, NSTISSC, Policy #11". July 2002. Available at http://niap.nist.gov/cc-scheme/nstissp_11.pdf

[16] "Partitioning Kernel Protection Profile, Preliminary Draft V0.3", *NSA C12*, October 2002.

[17] J. Rushby, "A Trusted Computing Base for Embedded Systems," *Proceedings of the 7th Department of Defense/NBS Computer Security Conference*, 1984, pp 294-311.

[18] "Definition" Available at http://www.webopedia.com/TERM/M/Moores_Law.html

[19] C. Weissman, "BLACKER: Security for the DDN, Examples of A1 Security Engineering", Presented at 1988 IEEE Symposium on Security and Privacy, *Proceedings Conference IEEE Symposium on Security and Privacy,* Oakland CA, 1992, pp 286. Available at http://www.computer.org/proceedings/sp/2825/28250286abs.htm

[20] D. E. Bell, and L. LaPadula, "Secure Computer Systems: Unified Exposition and Multics Interpretation", *Technical Report ESD-TR-75-306,* ESD/AFSC, Hanscom AFB, Bedford, MA, 1975. Available at http://csrc.nist.gov/publications/history/

[21] M. Taylor, "The Raw Prototype Design Document V4.11", *Department of Electrical Engineering, MIT,* 2002. Available at: http://www.cag.lcs.mit.edu/raw/documents/RawSpec99.pdf

[22] K. Mai, et al, "Smart Memories: A Modular Reconfigurable Architecture," *Computer Systems Laboratory, Stanford University,* 2000. Available at: http://mos.stanford.edu/papers/km_isca_00.pdf

[23] "Trusted Computing Platform Alliance Main Specification V1.1b," February 2002. Available at: http://www.trustedcomputing.org/

[24] B. Hashii, "Formal Specification Languages and Theorem Provers", *Northrop Grumman Corporation*, El Segundo, CA, December 2001

[25] D. Jackson, and J. M. Wing, "Lightweight Formal Methods", *IEEE Computer*, April 1996, pp 21-22

[26] B. Hashii, "Using Alloy to Formally Specify MLS-PCA Trusted Security Architecture", *Northrop Grumman Corporation*, El Segundo, CA, July 2003

[27] D. Jackson, "Micromodels of Software: Modeling & Analysis with Alloy", *MIT Lab for Computer Science,* November 2001. Available at http://sdg.lcs.mit.edu/alloy/book.pdf.

[28] *Grid Computing Center*. Available at http://www.gridcomputing.com/

[29] "Intelligent Software Agents Lab," Available at http://www.cs.cmu.edu/~softagents