Trustable Computing in Next-Generation Avionic Architectures

James O. Bondi

Defense Systems and Electronics Group
Texas Instruments
Plano, TX

Abstract

In tomorrow's "brilliant" weapons, next-generation avionic computers will need to orchestrate the actions of many subsystems while further maintaining the security of sensitive data, the integrity of key data and of system behavior, and often other key properties. Maintenance of these properties will help ensure that system execution is trustable, conforming to both prescribed policies and expected behavior.

For traditional security (confidentiality) as required by DoD, the policy to be maintained is well understood and essentially application independent. Although the weapon environment will surely render inadequate much of current security engineering practice and likely stress the security technology base, the now familiar, fundamental mechanisms of MAC and DAC will still form the foundation of suitable multi-level security (MLS) maintenance. For integrity maintenance, however, realistic policies are quite application dependent. In general, a specific integrity policy needs to comprehend not only certain behavioral aspects of the overall application, but also of potentially many distinct states within the application. Thus, integrity maintenance requires control derived from a state machine specifying "acceptable" application behavior.

1 Computing Properties that Need Trustable Enforcing

As the functions of modern weapons become more sophisticated, so too must the embedded electronics used to implement these functions. For flying weapons, the embedded aviation electronics, or avionics, are changing from yesterday's collections of independent, uncoordinated, special-purpose subsystems to orchestrations of cooperating, coordinated, intelligent subsystems. In modern avionics, this orchestration is performed by powerful control computers that integrate and direct the actions of many subsystems.
to produce, during flight, a new map, updated with respect to both terrain details and locations of potential enemy targets. After completing the updated map, the missile would return to base for "debriefing" and subsequent reuse.

1.1 Traditional Security (Confidentiality)

In the DoD or military context, security implies protecting sensitive information from unauthorized disclosure. Specifically, traditional DoD security policy mandates that suitable confidentiality of sensitive, classified data be maintained by limiting its accessibility to those persons with both sufficient security clearance and a specific need-to-know [DTC 85]. Some computer systems have been developed that successfully maintain this policy even while supporting users of various clearances and controlling their access to data of various classifications. Most of these multi-level secure (MLS) systems have been conventional, stand-alone uniprocessors performing non-life-critical data processing functions in a carefully controlled environment.

However, maintenance of traditional multi-level security (MLS) will also be required in unconventional avionic computing like that found in Figure 1-1's brilliant reconnaissance missile. Figure 1-2 shows plausible security classifications and security clearances for the key data and key processes, respectively, that interact to support the missile's function. Figure 1-2 suggests that classifications of available maps in the base computer's map library range from top secret (TS) for newer, recently refined maps to secret (S) for older, somewhat outdated maps. Figure 1-2 also suggests that raw RADAR, IR, and TV data coming from missile imaging sensors are classified just S. Thus, even in a normal mission scenario, a multi-level secure (MLS) computing environment must be maintained in the proposed missile to ensure that processes can only access "appropriate" data.

Specifically, in Figure 1-2, mandatory access controls (MAC) must ensure that the clearance level associated with each process dominates the classification of all data it observes, or reads, and that data remain sufficiently classified during subsequent manipulation. For example, MAC must ensure that, irrespective of process identity, an S process (like "Sensor Fuser") can never read TS data (like "Flight Control Map"). Additionally, discretionary access controls (DAC) must support further access restriction that is based on process (or associated user) identity. For example, DAC must be able to prohibit TS process "Pilot" (even though its clearance is sufficient) from reading TS data "Updated Map" simply because, according to Figure 1-2, this particular access is not wanted or needed.

For any complex weapon system like our hypothetical missile, several special operating environments or situations can occur in which the harm possible from security breaches is potentially even more severe [Jo 89]. For example, consider missile development, when multiple contractors must share a prototype missile for software development, yet must be constrained to access only data for which each has approval and need. Also, consider missile capture, when the missile must resist dissemination of critical information to the enemy even under intense enemy probing. Even in such environments, the MAC and DAC mechanisms alluded to above will be required as the foundation of suitable MLS protection.

We note, however, that DoD security policy is fundamentally "fixed" in nature. It is a simple generic policy which remains immutable even as the sets of programs and data under its control may vary dynamically. DoD security policy and its supporting mechanisms do not comprehend adaptation to different states within an application, or even to different applications. Adaptation, if any, to special situations like those cited above is handled outside the traditional DoD security maintenance controls.

1.2 Integrity

Traditional security controls maintain the confidentiality of sensitive data by guarding against undesired outward flows of such data into unsuitable hands. On the other hand, integrity controls maintain at least the "uncorruptedness," or "correctness," of critical data by guarding against undesired inward flows into such data. Intuitively, maintaining the correctness of a body of data requires more than simply controlling the actual inward flows of any new data into that body. It also requires ensuring that only data of proper "pedigree," produced from approved inward flows to alter key existing data [Wi 90]. This paper thus proposes a rather liberal definition of integrity that implies maintaining both the correctness of key system data and the correctness of system computational behavior.

Not surprisingly, exactly what constitutes correctness of key data and correctness of program behavior is typically application dependent. Therefore, effective and realistic integrity maintenance policy, unlike security maintenance policy, is generally quite application dependent. In fact, the details of integrity policy often further depend on, and vary with, specific states of the application. Put simply, integrity maintenance requires "controlling, for each application, who can do what to whom in which states."
Figure 1-3 illustrates the state-dependent nature of integrity maintenance within our hypothetical brilliant reconnaissance missile application. Figure 1-3 suggests the existence of at least three application states, each distinctly constraining allowed data accesses in a fashion consistent with high integrity execution of the overall application. For example, consider the following allowed manipulations of data "Flight Control Map" suggested in Figure 1-3. First, in state "pre-mission," only process "Librarian" is allowed to write to data "Flight Control Map," and no process may read this still forming map. Then, in state "during mission," no process may write this now fully formed, critical map, and only process "Pilot" may read it. Finally, in state "post-mission," any process could conceivably be allowed to write this no longer needed map.

Clearly, realistic integrity policy is not "fixed" in the sense described above for DoD security policy but, instead, may vary in many dimensions. Realistic integrity policy is specifically adapted to the associated application and, furthermore, may even need to adapt dynamically to distinct states within the application. Integrity maintenance controls must properly support such adaptation, both static and dynamic [Jo 91].

1.3 Other Special Properties

Given the often critical nature of a brilliant weapon's mission, maintenance of other properties beyond security and integrity will typically merit serious attention. This is especially true for various correctness properties [SSH 85], [HAS 91] including safety, virus resistance, and fault-tolerance. Even though these properties receive little explicit attention in this paper, it is encouraging to note that the integrity controls proposed herein naturally provide some support of them. For other crucial properties, like real-time-assured performance, it is not clear that the proposed controls naturally provide any particular help or hindrance.

![Diagram: Avionic Computing in a "Brilliant" Reconnaissance Missile](image-url)
Fig. 1-2 Traditional Security Maintenance - Its Static Nature

Fig. 1-3 Integrity Maintenance - Its Dynamic Nature
2 Traditional Security (Confidentiality) Maintenance

2.1 Security Policy Characteristics

Traditional DoD security policy is fairly simplistic and quite static in nature. It is simplistic because its fundamental properties are few in number and easily expressed in concise mathematics. It is static because none of its access decisions vary with system, or application, state. As implied previously, MAC decisions can be derived from comparing simple, essentially fixed labels attached to both processes (users) and target units of data. DAC decisions go beyond such labels to compare the identities of processes (users) and target data units but remain invariant to system, or application, state.

2.2 Security Model & Policy Enforcement

To enforce traditional DoD security, we can use the now familiar Bell-LaPadula (BLP) model [BLP 75]. In the BLP model, programs (executing on behalf of users) are termed subjects. Groups of data available for potential access are termed objects. The access of subjects to objects is constrained according to both the mandatory (clearance/classification-based) and the discretionary (user-specifiable) components of DoD security policy.

2.2.1 Mandatory Access Control

MAC enforcement is derived from tamper-proof labels "attached to" every subject and every object (Figure 2-1). Such labels permit easy comparison of subject clearance "level" versus object classification "level" to assess dominance. On read attempts, MAC insists that the subject level be dominant, enforcing the simple security property. On write attempts, typical MAC implementations insist that the object level be dominant, enforcing a somewhat restrictive version of the so-called *-property. By enforcing both properties, MAC ensures that data can only be accessed by those with sufficient privilege.

2.2.2 Discretionary Access Control

DAC enforcement is conceptually derived from a user-modifiable access matrix like that of Figure 2-2. (Equivalent structures, like an individual access control list associated with each object, can be used instead of a single access matrix.) For each attempted subject-to-object access, DAC insists that the corresponding matrix entry allow the requested access mode, thus enforcing the discretionary security property. Via DAC, users may, for objects they "manage," further restrict the access otherwise permitted by MAC.

2.3 Security Architecture

It is now accepted practice to implement security architectures as a series of hierarchical layers. Figure 2-3 illustrates that, within the overall operating system supporting traditional security, the critical functions of MAC and DAC are often isolated within a trusted security kernel situated directly atop the hardware, beneath other operating system services [Br 87], [BT 89]. By thus isolating security-critical functions within a distinct and small kernel, their correct implementation is more easily demonstrated.

---

Fig. 2-1 Labels for Mandatory Access Control (MAC)

Fig. 2-2 Access Matrix for Discretionary Access Control (DAC)
3 Adding Integrity Maintenance

3.1 Integrity Policy Characteristics

Compared to traditional security policy, effective integrity policy is generally more sophisticated and more dynamic in nature. It is more sophisticated because its rule set tends to be application-dependent, complex, irregular, and not easily expressed in concise mathematics. It is more dynamic because it is usually even application-state-dependent. Effective integrity policy maintenance cannot, in general, be derived from just traditional MLS MAC-like and DAC-like mechanisms. As with traditional MAC, key integrity control decisions must be mandatorily enforced (without room for user discretion). However, realistic integrity control will usually require more policy flexibility than the simple rules and fixed labels of traditional MAC can support [ALW 92], [Lee 88], [Sh 88]. In general, realistic integrity control must comprehend, for each supported application, perhaps intricate, state-machine-like specification of what actions the various application processes can do to the various application data groups from within each distinct application state.

From the above discussion, it is clear that integrity maintenance as proposed by Biba [Bi 77], with its MAC-like use of simple integrity labels, is generally inadequate. The debate-provoking and research-inspiring integrity maintenance proposed by Clark-Wilson [CW 87], [ALW 92], with its more sophisticated control of "what subjects can access/manipulate what objects using which approved functions," made major strides forward. Likewise, LOCK, with support of limited execution-monitoring state machines (called "assured pipelines"), helped advance the state of integrity maintenance [SBL 87], [SBL 89]. However, we contend that realistic integrity maintenance must often go even further to also comprehend potentially many distinct application states, varying its control decisions according to application state. Only by explicit recognition of state can okay/denial of each requested operation be efficiently tied to what history of operations have already been performed.

3.2 Integrity Model & Policy Enforcement

Integrity control with the required sophistication and flexibility must come from a control structure that functions as a state machine. Such an integrity state machine (ISM) specifies high-integrity behavior of the overall application by specifying its allowed behavior in each state. Specified behavior includes at least the allowed subject-to-object access patterns but can conceivably be far more detailed. With such an ISM, the mission-state-varying accesses of "Flight Control Map" suggested in Figure 1-3 can be readily accommodated.

To help ensure the effectiveness of the ISM in catching "undesired," low-integrity application behavior, the ISM cannot be simply derived from the detailed application implementation. Instead, the ISM must be developed independently of the actual application implementation so that it can realistically police the executing application. Such independent policing can even help enforce various correctness properties, like computing "safety," and, through detection of imposter applications, can provide some measure of "virus resistance." It is conceivable that the ISM should be developed first, by some trusted authority, and serve as an unalterable, presumed-correct specification that guides all other aspects of ultimate application development. (This development approach would allow some well advised "separation of duty" among the various system development phases [CW 87].)

3.2.1 Mandatory Behavior Control

An integrity state machine (ISM) as discussed above can constrain an application to high-integrity execution by effecting mandatory behavior control (MBC) as depicted in Figure 3-1. Each state of the ISM conceptually contains a matrix that, like a DAC matrix, comprehends all system subjects and objects. Unlike a DAC matrix, however, each element of such a state-contained matrix specifies not only allowed subject-to-object access modes, but also potentially more detailed behavior constraints such as allowed data pedigrees for okay of data stores. Also, unlike a DAC matrix, each element provides the next-state specifications needed to guide ISM state-change
operations. Finally, unlike a DAC matrix, no matrix elements are alterable at user discretion. Instead, all must be used, as provided, to constrain application execution in mandatory fashion. When the ISM is invoked for MBC, the matrix in the current state is consulted to okay or refuse attempted accesses and potentially other behavioral details. If the requested actions are okayed by the current state, no state change is needed. If the requested actions are not okayed, however, the ISM will move either to a next state that understands and okay the attempted actions, or will move to a special error-handling state to initiate appropriate remedial action. Besides error-specific recovery tactics, such actions might include capture of relevant system state for subsequent audit and diagnosis.

It is expected that, for a great many applications, realistic integrity maintenance can be derived from an ISM whose states are distinguished primarily by differences in the application's object accessing needs (as opposed to differences in even more detailed execution behavior patterns). Fortunately, such an ISM can provide up to moderate-grain integrity control with little, if any, special hardware required. Address faults stemming from the somewhat extended virtual addresses already supported in some machines, like the MIPS Computer Systems R4000, for example, can trigger ISM activity to effect appropriate MBC. The address faults must come, upon attempted object access, when either the basic virtual address or when the integrity state, as an extension to the basic address, mismatches against available virtual-to-physical mappings. (Suitable virtual addressing support is discussed more in Section 4.)

3.3 Integrity Architecture

Like the critical security functions of MAC and DAC, the critical integrity function of MBC is also appropriate for incorporation in a compact, layered kernel. Figure 3-2 suggests that MBC be provided by a new kernel layer added between the more conventional MAC and DAC layers to yield an architecture providing integrity combined with security. In such an architecture, the integrity/security kernel could invoke DAC, then MBC, and then MAC, okaying requested accesses only when all three checks are passed.

![Integrity State Machine (ISM) for Mandatory Behavior Control (MBC)](image-url)
4 Recommended HW/SW Support

It appears that hardware and software (HW/SW) support required for integrity maintenance, via the proposed MBC, is compatible with that commonly used for traditional security (confidentiality) maintenance. Thus, integrity-specific support can progressively complement a traditional security base as required to yield an effective integrity/security platform.

4.1 Traditional Security Base

HW/SW support considered essential to a traditional security base is well understood. In fact, due to their usefulness beyond strictly security support, many of these base features are becoming commonplace. In the area of SW support, the kernelized SW architecture of Figure 2-3 is utilized to cleanly encapsulate security critical functions in small, distinct layers where their functionality can be shown correct with high assurance. In the area of HW support, features recommended long ago by Tangney [Ta 78] are still relevant and now common on most machines because of their universal utility. These are

- multiple, distinct privilege states, and
- (virtual) memory management mechanisms that
  can, for entities sharing main memory,
  - isolate them from one another and
  - limit the accessibility of each to specified modes (like "read-only").

If a fault-induced trap is added to Tangney’s other recommendations, a really strong foundation for kernelized system architectures results. The distinct privilege states effectively stratify machine functions, and the SW composed of them, into a privilege hierarchy of layers. When coupled with a trap mechanism that, upon certain “fault” conditions, forces transfer of program control to a more privileged SW layer (“lower” layer in Figure 2-3), such privilege states enable more privileged SW layers to control, or constrain, less privileged layers. For example, an application’s attempt to execute any unsuitably privileged instruction would elicit a “privilege fault” and forcibly invoke reaction from some lower layer in the operating system. Similarly, an application’s attempt to first reference an object, before memory management HW had established a virtual-to-physical mapping, would elicit a “virtual address fault” and forcibly invoke an accessibility ruling from the DAC, and maybe also MAC, layer(s) within the kernel.

4.2 Adding Moderate Integrity

With straightforward HW/SW extensions to the above described security base, a significant degree of integrity support can be realized together with security support. In the area of SW support, the extended kernelized SW architecture of Figure 3-2, with an integrity-specific MBC layer added, is recommended. In the area of HW support, a flexible trap based on an extended virtual address (EVA) fault can efficiently recognize a variety of points at which the new MBC layer should be invoked to police application integrity.

As shown in Figure 4-1, an EVA is formed by concatenating several extensions to a basic virtual address that points to a page (or other hardware-managed unit of virtual address space). The extensions permit flexible characterization of an application’s use of the basic address, specifying associated details such as the access mode, the application state, the pedigree indicator (on data stores), and even other behavioral details. With EVA, an instruction-generated address mismatches against an established virtual-to-physical mapping if any of the extensions or the basic virtual address mismatch. Thus, an EVA fault can invoke MBC integrity checks not only when fundamentally new addresses are generated but also when attempted use of an address differs from uses already okayed in the recorded virtual-to-physical mappings.

Note that minor degrees of virtual address extension, especially the "access mode" specifier needed in a traditional security base, are already available in most modern processors. Furthermore, some new processors, like the MIPS Computer Systems R4000, now provide an “application ID/state” field that could coordinate application
execution with a constraining ISM to cost-effectively yield moderate-grain integrity control. 

Thus, with minimal EVA now appearing and requisite kernelized software architecture well understood, fundamental HW/SW support for integrity maintenance is at hand. Given sufficient attention to compaction of ISM representation and similar implementation issues, practical systems enforcing up to moderate-grain integrity are realizable.

Fig. 4-1 Extended Virtual Addressing (EVA) Support

### 4.3 Toward Finer Integrity

With suitable enhancements to the minimal integrity support HW discussed above, it appears that ISM-derived MBC could be extended to enforce fairly fine-grain integrity. For instance, special but conceptually simple processor HW could be added to track the pedigree of computed data. It made part of the EVA, a "pedigree indicator" could be checked against established virtual-to-physical mappings to trigger pedigree-based ISM rulings upon attempted data stores. In fact, as Figure 4-1 implies, the flexible EVA mechanism could accommodate many "other" extensions that, like data pedigree, support finer and finer monitoring of application execution activity. For support of ultra-fine-grain integrity control, it is conceivable that even detailed "execution signatures," indicative of detailed instruction/data sequences, could form EVA fields that are checked against expected signatures.

### 5 References

- [FGS 89] "... Fine-Grained Secure Computing," Ti Proposal 27-R89 (to DARPA), Computer Science Center, Texas Instruments, Dallas, TX, May 1989.


