Ganesha Bhaskara, Timothy E. Levin, Thuy D. Nguyen, Cynthia E. Irvine, Terry V. Benzel, Jeffrey S. Dwoskin, and Ruby B. Lee. Virtualization of a Processor-based Crypto-Protection Mechanism and Integration within a Separation Kernel Architecture. Princeton University Department of Electrical Engineering Technical Report CE-L2006-006, November 2006.

# Virtualization of a Processor-based Crypto-Protection Mechanism and Integration within a Separation Kernel Architecture

Ganesha Bhaskara[§], Timothy E. Levin[*], Thuy D. Nguyen[*], Cynthia E. Irvine[*], Terry V. Benzel[§], Jeffrey Dwoskin[₪], Ruby Lee[₪]

***Abstract***: *This paper describes the design of an integrated high assurance separation kernel and a Secret Protected (SP) hardware for cryptographic services. Integrating SP with the separation kernel requires (1) augmenting the SP instruction set with additional hardware instructions to aid virtualization and ensure that the confidentiality of user's secrets are protected to the same extent as in the original design of SP (2) augmenting the separation kernel to ensure minimization of information flow via covert channels resulting from integration of SP (3) reconciling the user specific model and the usage model of the integrated design and (4) controlling flow of information about user's secrets across the different Secrecy, and Integrity labels. The architecture called Secure Core, is designed for networked mobile devices to be used by a single user at a time. We define usage scenarios in which users may need to assume different roles, that translate into different security profiles for the user. We begin with a description of the separation kernel and the SP architecture. This is followed by a description of the hardware requirements for virtualization of SP integration and use of the virtualized cryptographic SP services. We find that the main changes required to SP are the ability to save and restore SP state as the SecureCore kernel switches between virtual machines, so that isolation properties are maintained when using SP. We conclude with a security analysis, including the effect of the virtualization and integration on the confidentiality/integrity of user secrets as well as enforcement of MAC on user secrets like cryptographic keys. The integration of SP hardware-based secure cryptographic services with the separation kernel software architecture provides essential isolation, confinement, key-management and cryptographic services, forming a strong Secure Core as a basis for future secure PDAs, and a rich environment for secure system research.*

# 1 Introduction

Today's fast paced highly networked digital world places particular emphasis on a variety of new computing and communication devices that are employed in critical applications. This rapid evolution of computing platforms from stand-alone desktop systems to highly-networked pervasive mobile computing devices magnifies the vulnerabilities in current systems, and the need to establish more trustworthy computing and communications platforms. A typical example of such a device is a networked mobile device like a PDA, designed to be used by a single user at a time. A user, for example, a first responder or a soldier, may need to assume different roles (or security profiles) at different points in time to access data and services at different levels of integrity and/or secrecy. For example, to access data on the web, the user is required to switch to low integrity, low secrecy security profile where as to access secure documents like medical records or design plans of buildings, the user may be required to switch to a security profile of higher level of integrity and/or secrecy.

Although virtual machine monitor (VMM) implementations such as (Zen [barh03] or Vmware [devi98] ) might appear at first glance to be suitable for such systems, they lack sufficient assurance for highly critical applications, such as are those required for systems similar to DoD's Global Information Grid (GIG) and the Navy's FORCEnet. We have chosen what we believe is a conceptually simpler separation-kernel (SK) [rush82] based approach instead of the full-featured VMM approach, as the SK will allow derived systems to be more accessible to analysis and understanding, and on that basis alone have the potential for higher assurance.

In addition to being aligned with the current investigative and developmental efforts pursued by NSA and the military services [vanf05], the SecureCore [irvi05] approach of building a relatively simple layer of Secure Services (SS) on top of a high assurance separation kernel (SK) to service the OS promises to provide several advantages (see Illustration 1). For one, the use of a high-assurance SK provides a foundation for a high assurance system, whereas we are not aware of any approach through which a system built upon a low assurance commodity VMM could achieve anything higher than (that same) low assurance. Secondly, a VMM typically limits inter-OS traffic to the network interface, which can prevent the system engineer from using the most natural, and potentially

§ University of Southern California/Information Science Institute, * Naval Postgraduate School, ₪ Princeton University

simpler, abstractions for sharing; whereas with an SK, various resources in addition to the network interface can be shared between OSs. Finally, the goal of SS in SecureCore is to provide a translation of the separation kernel's exported resources such that they can be easily used by a simple hand-held operating system. This is in contrast to the much more difficult requirement for a general purpose VMM to support many different workstation-class OSs, without modification. In short, the idea is to provide simple, flexible interfaces, high integrity user cryptographic features, and high integrity trusted functions for separation kernel based high assurance systems.

The Secure Core architecture is designed for networked mobile devices that are used by a single user at a time. In addition to the controlled sharing between security domains related to different user security profiles provided by a Separation Kernel, the protection of user-generated sensitive information stored on the device and potentially accessed via public networks is essential for the user of a secure PDA. Such information can be protected by cryptographic methods, but a common problem for users, regardless of whether their system is based on a monolithic OS, VMM or SK is the lack of flexible, trusted access to encryption services. As long as untrusted OS or VMM software is part of the processing chain for accessing user keys or for using them to transform data, the results can only be as secure as that of the untrusted software. Keys that are permanently burned into ROM may offer relatively higher assurance, but this penalizes the user by locking him to the device, and making it difficult to change keys. Thus, a wireless, mobile, hand-held device requires a ***secure***, ***portable*** and ***convenient*** way to provide cryptographic services, including storage of user's secrets like cryptographic keys. The Secret Protected (SP) [rlee05a] [rlee05b] architecture defines a minimalist set of processor hardware features, which can be added to any microprocessor, that protect a user's master keys enabling secure cryptographic services even in the presence of an untrusted commodity OS. The integration of SP hardware-based secure cryptographic services with the SK based software architecture provides essential isolation, confinement, key-management and cryptographic services, forming a strong Secure Core as a basis for future secure PDAs, and a rich environment for secure system research.

This paper describes the design for integrating SP protection services with a high assurance separation kernel. The native SP design assumes a single user with a single security profile. The SecureCore usage scenario supported by the SK requires multiple user profiles and related security domains (e.g., levels). Thus, this integration requires: (1) augmenting the SP instruction set with hardware instructions to virtualize SP state between security domains, (2) augmenting SK to ensure covert channels do not result from integration of SP and (3) providing access control for managing SP-related user secrets across the different security domains.

This paper is organized as follows: In Section 2, we describe the separation kernel and the SP architecture. This is followed – in Section 3 – by a description of the hardware requirements for the virtualization of SP and for the use of SP's virtualized cryptographic services. We find that the main changes required to SP are related to the saving and restoring of SP state as the SecureCore kernel switches between virtual machines; secure swapping of SP state ensures that isolation properties are maintained when using SP in a virtualized environment. In Section 4, we present a security analysis which discusses the effect of integrating and virtualizing SP on the confidentiality and integrity of user secrets, as well as on the enforcement of Mandatory Access Control (MAC) on user secrets – e.g. cryptographic keys.

# 2 Overview of SK and SP in SecureCore

In this section we present the two building blocks of the SecureCore architecture, namely the Separation Kernel and the Secret Protected architecture with emphasis on aspects relevant to this paper.

## *2.1 Overview of the Separation Kernel Architecture*

The target SecureCore architecture is based on a thin MLS-aware Trusted Management Layer (TML) sitting on top of a hardware platform with the security-aware processor features of SP. The TML consists of the Separation Kernel (SK) and the Secure Services layer (SS), as shown in Illustration 1. The SK resides in the most privileged ring and utilizes hardware mechanisms/instructions to provide isolation between blocks (as in [inte05], [adva05]) and between processes/tasks within a block. The SK is the only software component which directly interacts with the hardware, so it can partition the hardware resources into blocks[1] blocks and control the flow of information

[1] "Partition" is sometimes used to refer to the blocks of a partition, but can cause confusion, so that term is not used here.

between them. For this paper, we assume that the processor has the usual architecture with 4 positive privilege rings, enhanced with 2 negative privilege rings. We refer to the negative rings as *Virtual Machine Monitor (VMM) [gold72] or TML privilege* domains. The following table shows the allocation of SecureCore components to the different privilege rings.

*Table 1. Allocation of SecureCore components to privilege rings*

| ***Ring*** | ***Ring (-2)*** | ***Ring (-1)*** | ***Ring (0)*** | ***Ring (1+)*** |
|---|---|---|---|---|
| **SC Layer** | SK | SS | OS | Applications |
| **Block Access Privileges** | All blocks with no restrictions | All blocks with restrictions on operations | Single block with no restrictions within that block | Single block and restricted to single task / address space |

Resources are allocated to blocks at the time of initialization or startup. The SS exports the partitioned resources as abstractions that the operating systems in each block can utilize. It provides secure virtualized networking, user I/O, storage, access control and cryptographic services. In this paper, an SK managed "block" may be understood to have the functionality of a "Virtual Machine", and TML to have the functionality of a trusted VMM, although there are significant differences compared to existing VMMs like Xen [barh03] or VMware [devi98] such as TML-controlled "read-down" of resources between blocks. SS assigns one confidentiality-integrity-label pair per block, where each label may contain both a hierarchical sensitivity level and a non-hierarchical category indicator. Illustration 1 shows the prototype design with two modes of operation: normal and trusted. In the normal mode (low integrity), a simple commercial OS hosts off-the-shelf applications. The trusted blocks can be assigned a label "range," within which the labels of all of its processes and other resources must be encompassed. In the trusted mode, special high assurance applications run on a high assurance resource manager, the TOS. The trusted mode

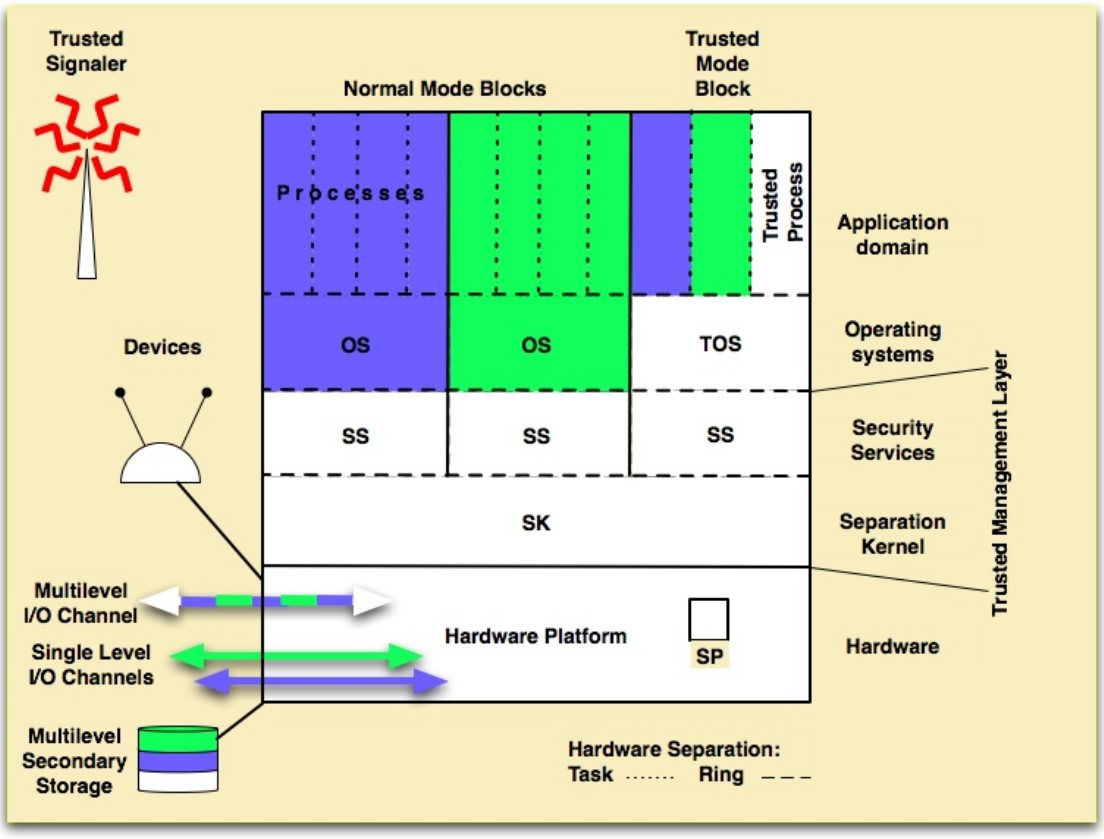


*Illustration 1: SecureCore system architecture*

provides an environment to manage transactions that must result in data of high *semantic* and *syntactic integrity* (see Section 2.2). Since all components in all execution paths in trusted mode have high semantic integrity, and the

trusted mode components are protected by the TML from corruption by untrusted blocks, the outputs from the trusted mode session will have high integrity with respect to user inputs. The TML enforces an MLS policy that allows read down, controlled write up with respect to secrecy and supports asynchronous notifications among blocks. Arbitrary write up is in general, not allowed but may be supported for certain encapsulated objects, such as keychains and eventcounts [reed79]. For example eventcounts can be used to signal a high sensitivity level subject to read down. For example, a subject labeled low secrecy and low integrity with data for a high secrecy and low integrity subject, can use an event count to signal the availability of data to be read down.

Each normal mode "guest" OS operates within a block, which means that it is a single-level entity. However, the OSs are not isolated, as is the case with a classic virtual machine that only allows inter-block flows via network devices. Inter-block resource accesses are allowed, but must conform to the MLS policy. For example, an OS or one of its processes can read resources in any block that has a confidentiality label equal to or less than its own. A guest OS can be configured to restrict its processes to a subset of the accesses allowed to the OS. The multiplicity of blocks and sensitivity levels of information that must be kept separate motivates the virtualization of SP services, discussed below.

Context switches between blocks are user- and process-driven. The user is active in one block at a time, and if there are no background processes in the other blocks, the user's block remains active until the user logs out or changes to another block. If background processes are running in other blocks when the user is logged in, then all active blocks will be scheduled with fixed time slices to prevent covert timing channels that could be exploited by malicious applications like Trojan horses.

When a user wants to change security/integrity levels or otherwise utilize services from a different block than the current one, the user presses the secure attention key (an external non-spoofable physical input). The SK services this interrupt and passes control to a trusted process in the Secure OS block. This trusted process interacts with the user to negotiate access to a user session in a block with the desired security and integrity attributes.

## *2.2 Overview of Native SP*

The objective of the Secret-Protected (SP) architecture [rlee05a] is a design for a very small set of new processor features that can be incorporated into a broad range of processors. SP enables secure and convenient protection of the user's critical secrets as well as protection of critical code segments, with minimal changes to hardware and software. SP protects the user's secrets from various hardware and software threats that could compromise the confidentiality or integrity of secret or sensitive information. SP also provides portable access to user secrets across all SP-enabled devices by avoiding the use of permanent factory installed master keys. A User Master Key (UMK) is entered directly by the user into the volatile UMK register through a secure I/O channel. The user protects secrets with the UMK by first loading the UMK with an SP instruction. The presence of the UMK in the SP device then enables secure access to any number of other keys, passwords and secrets. These protected keys can be used to performing cryptographic transformations on other protected data, code or files in a secure way. A device-specific Device Master Key (DMK) is used to protect *trusted software modules* (TSMs) which are the only code allowed to access the UMK or SP management functions. The TSM instructions are protected by keyed hashes over instruction cache lines, which are generated using the DMK at TSM installation time. The hashed values are stored inline with the code. Since SP instructions are a part of the processor ISA, SP also reduces to an insignificant level performance concerns associated with the traditional secure co-processor architectures.

Before we introduce the native design of SP in more detail, we introduce the following terminology:

- *Semantic Integrity* in the context of cryptography refers to the association of the correct key with a particular intended use (e.g., an application or service). An example is the key associated with a user's email account.
- *Syntactic integrity* refers to the soundness of a key relative to its initial or expected value. For example, a corrupted key has low syntactic integrity

Protection of the semantic integrity of a user's secret such as a key or a key chain is an application and system design problem. An SP-enabled system depends on the applications, the individual TSMs, or their execution environments to ensure the right key is used, and to control access to TSM interfaces and encrypted keys, if those

services are required. Thus, an attacker cannot observe, or modify without being detected, the bits of any key protected by SP. The only way SP-protected keys can be observed, modified or used are through execution of the Trusted Software Module (TSM), and whose syntactic integrity is continuously and dynamically verified by automatic SP hardware mechanisms for code integrity checking. An important aspect of SP is that the UMK is associated with the device until the user overwrites it. For another user to use the device or for the same user to use the device to access data encrypted with a different UMK, the new UMK must be entered. This property of SP is ideally suited for the usage model of PDA-like devices. Of course, a UMK change must be done with care as previously encrypted data will not be accessible until its enabling UMK is again loaded.

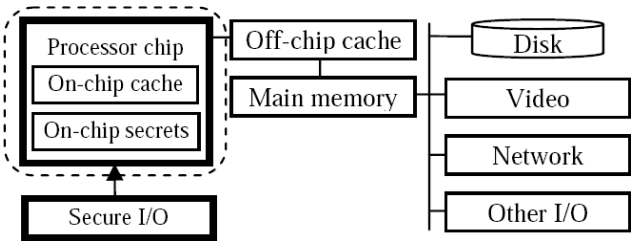


*Illustration 2: Trust boundaries(indicated by dark boarders) in SP architecture*

SP protects use of TSMs through its Concealed Execution Mode (CEM). Before a TSM interface can be invoked by a process, a CEM session must be started by executing the *begin_cem* instruction. At the end of the CEM session the *end_cem* instruction must be executed to switch the processor back to normal mode. Thus, a TSM is executing *if and only if* SP is in a "CEM session." The *begin_cem* and *end_cem* instructions toggle the *in-CEM* bit in the CEM Status register (See Illustration 3). Once CEM is started, the SP processor continuously performs on-the-fly checks for the instructions' integrity with respect to the hash of instruction cache lines, until the CEM session ends. Any security critical data that needs to be moved outside the processor's trust boundary (indicated by the dark borders in Illustration 2) during TSM execution in CEM mode is also hashed and encrypted using the DMK. If an interrupt occurs when the processor is in CEM, SP encrypts the contents of all the general purpose registers (in place), and then calculates the collective hash of them. This *interrupt hash value* and the interrupt return address are stored in special SP registers that are inaccessible to software. Only then does the SP processor transfer control to the operating system to service the interrupt. Hence, SP ensures

- the integrity of the return address and the contents of the general purpose registers, and the
- the confidentiality of the contents of the general purpose registers, even in the presence of a corrupted untrusted OS, preventing register replay and incorrect interrupt resume attacks.

A usage model [rlee05a] for protecting user data with SP is that of keychain construction and access. User keys are organized in a logical "key chain". The UMK is the root key for a chained encryption of the keys. If a user wants to use one of the keys to perform a cryptographic operation, then the keys on the path to that key are first decrypted using the UMK and the subsequent keys along the path, and the selected key is used to perform the required operations. The user data in this tree is used as keys to encrypt and hash data that is lower in the tree. Thus, by protecting the UMK, the entire key chain is protected for both confidentiality and integrity .

# 3 Integrating SP in a Separation Kernel Architecture

There are several problems associated with the integration of SP with the Separation Kernel architecture. First, the single CEM execution-thread model of SP must be reconciled with that of the "multitasking" of blocks supported by the SK architecture. Second, the secrets a user has access to, may have different MLS confidentiality and integrity labels. Third, the shared SP hardware or software resources must not give rise to covert channels. The aim of the integration effort is to add the the hardware based protection of user's secrets provided by SP to the high assurance SK to obtain an architecture with the high assurance properties of both the components.

- ***Multitasking of CEM threads***: Though SP supports a usage model featuring multiple simultaneous TSMs in a given block, we focus on supporting only one TSM per block in this paper, to simplify the exposition. The problem of supporting multitasking of CEM reduces to virtualizing the SP hardware resources

between blocks. A single CEM thread at any time within a block is assumed for simplicity of exposition, but CEM multitasking can easily be extended within a block. The integrated SP-SK architecture must ensure that no block can lock shared resources, such as the CEM, from access by other blocks.

- ***Access control***: SP does not provide access control for the interfaces of the TSM or to user's encrypted data. Integrating the SP architecture into an MLS system needs to meet the following requirements
  - → Integration should have a minimal impact on the complexity of SK: otherwise, a reduction in the assurance of the SK could result
  - → Integration should not require MLS policies to be enforced outside of the SK: SK should not depend on external software or hardware components to enforce MAC policies.
- ***Covert Channels***: Since SP hardware and the UMK is shared amongst the blocks in the SK architecture, and high assurance systems may require the closure of covert storage channels it is important to prevent covert channels that may arise through the integration of SP cryptographic services. For example, the integrated SP-SK architecture must ensure that no block can even detect if another block is using SP services. If this were allowed, a High block could signal a "1" to a Low block if the Low block finds the SP services are being used, and a "0" if the Low block finds the SP services are not being used.

## *3.1 Multitasking of CEM threads*

When the SK interrupts one block and schedules another, it saves the complete state of the processor, including the state of the current task in that block, before restoring the state of the next block. As with multitasking in a normal secure OS, this is done such that the state of the current block does not affect the operation of the next block, for example, by the fact that a task in the previous block was using a given resource. In particular, the TML must be able to virtualize the state of the CEM in SP. The SecureCore solution is to provides new privileged instructions to store and load the SP state.

### *3.1.1 Analysis of SP state for virtualization*

Since no software -- even the TML -- can modify the UMK and DMK during normal runtime, they cannot be used to transfer information between blocks. This is a major SP virtualization decision that does not detract from SP security, while achieving transparent virtualization for blocks, free from covert channels.

The remaining SP state is the CEM state, which consists of the CEM status register (2 bits), the CEM Interrupt Hash register (128 bits) and the CEM Interrupt Address register (64 bits). In native SP, the two bits that indicate the status of the CEM act as a hardware lock that ensures that only one CEM session can be in active or interrupted mode at any given time i.e., the set of TSMs on the system, as a whole, are effectively an interruptible *critical section*. When a task is restored on a context switch and *CEM-interrupted* is set, after the new context is loaded, SP checks the new PC address against the value in the CEM return address register; it checks the collective hash of all the new general purpose registers against the hash of registers that was stored in the internal hash register, and if the hash matches, SP individually decrypts each general purpose register. The processor proceeds with the CEM mode if both the checks pass.

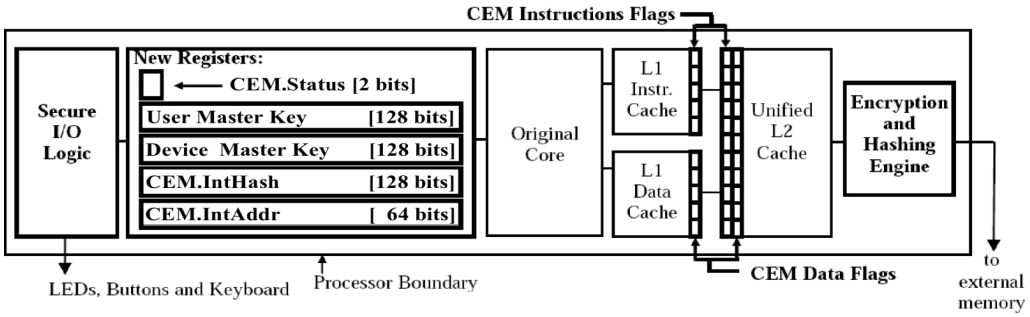


*Illustration 3: SP processor features*

These SP state registers must be virtualized between block switches by SK. Each block must be made completely unaware of the SP usage of any other block. There is also CEM data which is secure intermediate data belonging to a CEM thread executing a TSM. This is data in on-chip data cache lines tagged as "Secure Data" cache lines, and in the general purpose registers. This CEM data in the shared on-chip caches is not virtualized. However, since these caches are shared by blocks at different MLS levels, the SK must carefully handle cache partitioning to prevent covert channels between the different blocks. The CEM data in the general purpose registers are already protected by SP, if the block switch done by SK is considered a hardware interrupt; SP encypts and hashes the general-purpose registers, before passing control to SK.

While not a virtualization issue, residual information in data cache lines and the general-purpose registers may require the TML to clear both the data cache lines and the general-purpose registers when switching between blocks.

### *3.1.2 SP virtualization instructions*

Detailed analysis, tradeoff and design consideration resulted in two instructions to virtualize the state of SP: one to save and another to restore the CEM state registers of SP. The Save_SPstate instruction saves all CEM registers in ***one atomic operation***. Likewise, the Restore_SPstate instruction restores all CEM registers for a block in ***one atomic*** operation. The architectural decision to do this as an atomic operation in a single instruction is important to prevent attacks that try to interrupt the saving and restoring of SP state , thereby leaving inconsistent CEM state. Both these instructions may only be invoked from privilege level -2, which is reserved for the Trusted Separation Kernel.

*Save_SPstate* copies the CEM.IntHash, CEM.IntAddr and the CEM.Status registers to a specified address in memory. This memory location is specified by the SK, and should be in memory, only accessible to Ring -2.

*Restore_SP*state copies the CEM.IntHash, CEM.IntAddr and the CEM.Status registers back from a specified address in memory. These values should be retrieved from protected, Ring -2 memory so the SK can ensure that the SP state has not been not accessible to any VMs. Any data currently resident in the SP registers is overwritten when this instruction is used.

### *3.1.3 CEM state changes due to block switches*

Illustration 4 shows how CEM.Status ("CS") is intended to be managed when switching into a new block. Normal (CS=00) state means that no CEM thread is executing. This transitions to Active CEM state (CS=01) upon executing a Begin_CEM instruction. Upon a hardware interrupt or software exception, (e.g., during a block switch by the SK), the SP transitions CEM state from Active CEM to Interrupted CEM (CS=10) state. SP hardware allows the return to Active CEM state from Interrupted CEM only when the CEM.IntHash and CEM.IntAddr values match.

The Save_SPstate instruction can only occur out of Normal or Interrupted CEM states – and not out of Active CEM state. In either Normal or Interrupted CEM state, the Save_SPstate instruction stores CEM state of SP to the SP save area for the current block in ring -2 memory. This transitions SP into a new state called the CEM saved (CS=11) state. In this state, the SK may perform other operations associated with saving the state of the current block, block switching tasks and restoring or initializing the state of the new block. It then executes the Restore_SPstate instruction which restores the SP state of the new block. Note that the SK must initialize the SP state for each new block, so for example, CS properly initialized to Normal SP for a block invoked for the first time. Also, when the CEM state is restored after a switch from one block to another, either Normal or Interrupted CEM state is entered, depending on the next block's previous CS state..

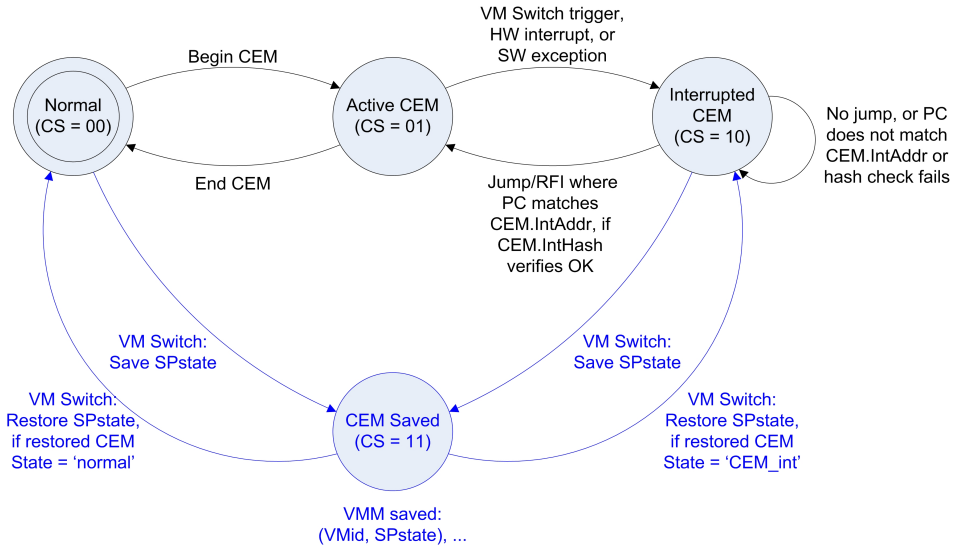


*Illustration 4: CEM state transition diagram*

### 3.1.4 TSM Privilege Allocation and CEM exceptions

The principle of layering [benzel05], [neum04], and [salt75] requires that a TSM be at least as privileged as the most privileged subject using it. If a TSM is used to encrypt and decrypt objects in Ring 0, which is allocated to the OS, the TSM is placed in Ring 0. In this case, access to the TSM from lesser-privileged rings is controlled by the OS – the applications call interfaces exported by the OS, which in turn uses the application's parameters to invoke the interfaces of TSM. The OS additionally performs TSM-related request queuing and dispatch, to provide an orderly execution environment for the applications. In general, the ring level of a TSM is dependent on the customer-site architecture or security policy for execution of that particular TSM. We note that other access control policies regarding the placement of the TSM with respect to rings, or TSMs independent of rings, are possible with the SP architecture [rlee05a], but SecureCore assumes a high assurance model and strictly hierarchical ring access controls.

Improper use of SP instructions will cause an exceptions to be raised by the processor. Exceptions raised while in CEM are treated like any other interrupt, triggering register protection before control is turned over to the interrupt handler. The two new SP instructions introduced for SP virtualization, Save_SPstate and Restore_SPstate, will raise an exception if issued in code not running in ring -2 (SK code) or in Active_CEM. [rlee05a] describes the other SP exceptions in detail.

## *3.2 Access control of SP-related User Data*

Integration of SP with the SK architecture requires reconciling the single user model of SP with the generic multi user model supported by generic SK architecture. In the SP architecture, when a user logs into the device and enters the UMK, that single key effectively becomes associated with each of the blocks on the device. However, since the guest OS maybe vulnerable to network attacks, there is a risk that a malicious remote process could access the UMK through the network interface. In the integrated architecture, the TML has the capability to prevent a remote user from connecting to a guest operating system. Such a restriction not only ensures that only the actions of the user in possession of the device can invoke commands directly or indirectly, but also ensures that the user's secrets entered by the user in possession of the device is not accessed by unauthorized remote users.

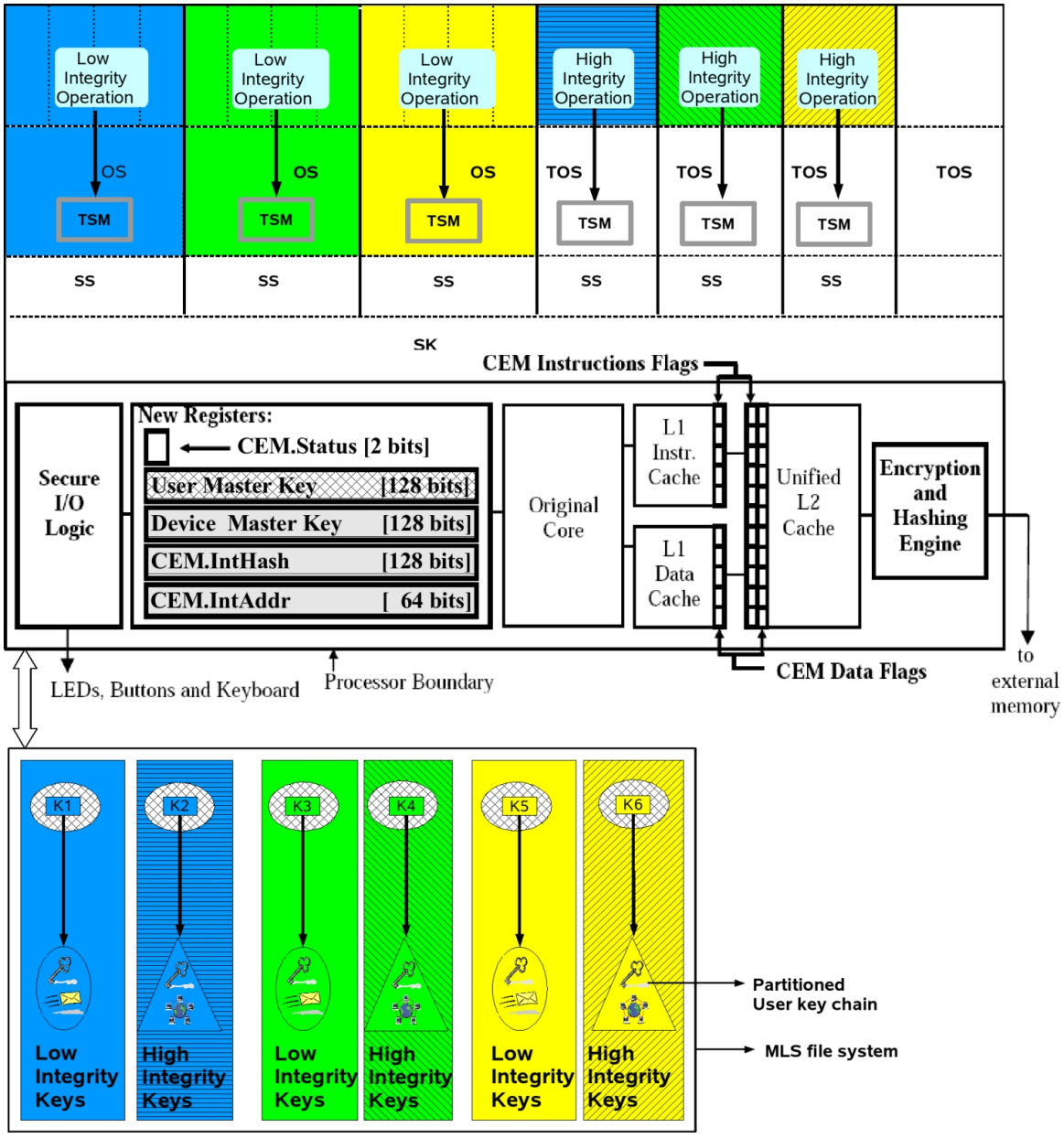


*Illustration 5: SP integration with SecureCore MLS architecture*

Keys and chains of keys are resources that can be shared by applications within a block as well as between blocks. Keys and key chains, likes files and directories in an MLS files system [irvi95], are stored in SK-managed resources according to their confidentiality and integrity levels, as shown in Illustration 5, and the MLS policy governs the access rights of different blocks to the key chains. Using this approach, i.e., a tree structure with increasing confidentiality and decreasing integrity going away from the root, keys can be managed without privileged or trusted processes and in a manner that is free of covert storage channels. Since an object's meta-data is explicitly stored in another object that is labeled at or above the confidentiality of the object, storage covert channels, which depend on unmanaged metadata, are prevented. Thus applications in each block will only be able to perform read and write operations on key chains if the MLS policies permit.

Since access to keys and key chains is controlled by the SK, the key chain managers need not be MLS aware and consequently need not be trusted with enforcing the MLS policy. In general, the integrity of a key chain manager needs to be at least as high as the integrity of the block that it services.

## 3.3 Addressing Covert Channels

The SK, SS and TOS are carefully designed so that on a block-level context switch, residual data from the previous block is not available to the user, OS or applications, if that availability would violate the MLS security policy.

Covert storage channels are eliminated by purging any state or data present in the caches in the processor before the state of the next block is restored, and by avoiding inter-block CEM locks. Timing channels are avoided by ensuring that the time for block context switching is made consistent by the SK, regardless of CEM activity. In this case, it is up to the SK to compensate (e.g., by inserting the appropriate delay) when a thread was not in CEM to prevent such a covert timing channel.

## 3.4 MLS Usage Scenarios

We describe some of the usage scenarios of the virtualized services of SP to show how the different functionality distributed across the hardware and different layers of the TML is used to achieve the intended security objectives.

As described in Section 1.1, SK can be configured to allow read down and controlled write up of exported resources with respect to confidentiality, and read up and controlled write down with respect to integrity. With these capabilities, read down and write up of encrypted data by the user can be permitted as a part of normal operation. Further, the enforcement of mandatory access control by the TML (SK and SS) ensures that objects that a subject cannot read are inaccessible to the subject and hence no operations can be performed on it. In this section, several scenarios illustrate how this is managed in the integrated SecureCore architecture.

In these examples, we assume that the existence of a kernel *k_append* operation for performing a "blind write" to a higher level. The application *A* has handles *A.k, A.m* for each key, *K,* or memory segment, *M,* in its address space. These are used as parameters in calls to services which provide the only access to those resources for *A*. In general, A.x indicates the only handle A has for accessing x. For keys, the handle can be as simple as the numerical index into the keychain, or can reflect the semantics of the application, such as a mail-signing key and a mail-encryption key. For the sake of simplicity, we will assume that: 1) the confidentiality level of the handle is the user's session level (2) the keys used for encryption are labeled the same as the data they encrypt, and (3) users are not allowed to use keys that are labeled at a higher confidentiality level than their session level.

In the scenarios shown in Table 3 , TSM operations resulting from calling the corresponding OS interface, are prefaced with "tsm_," SS operations with "ss_" and kernel operations with "k_". Buffers are labeled process-local memory segments, where the name of the buffer is its handle.

*Table 3: Usage scenarios*

| ***Scenario*** | ***Application A level:*** | ***Objects*** | ***Operations*** |
|---|---|---|---|
| 1: A user at a low confidentiality level wants to write encrypted data to a higher confidentiality level. | SECRET | Data *D* level: SECRET<br>Buffer tmp level: SECRET<br>Key *K* level: SECRET<br>Memory Segment *M* level : TOP SECRET (TS) | tmp := tsm_encrypt(A.d, A.k);<br>k_append(A.m, tmp); |
| 2: A user at a high confidentiality level wants to read encrypted data that is at a lower confidentiality level. | TOP SECRET | Encrpted Data $D^k$ level: SECRET<br>Buffer tmp level: TS<br>Key *K* level: SECRET | tmp := tsm_decrypt(A.$d^k$, A.k);<br>os_read(tmp); |
| 3: A trusted user employs a trustworthy program to downgrade data and encrypt it with a lower level key. | TOP SECRET | Data *D* level: TS<br>Buffer tmp level: TS<br>Key *K* level: SECRET | ss_downgrade(A.d, SECRET);<br>tsm_encrypt(A.d, A.k); |

| | | | |
|---|---|---|---|
| | | Memory Segment *M* level : TOP SECRET (TS) | |
| 4*: Scenario 4: A user at a low confidentiality level wants to append low level keys from the user's keychain to a high confidentiality level keychain. | SECRET | Key *K* level: SECRET<br>Keychain *C* level : TOP SECRET (TS) | ss_kchain_append(A.c, A.k) |
| 5: A user wants to create and append a key to her keychain. | SECRET | Key *K* level: SECRET (initially empty)<br>Buffer tmp level: SECRET<br>Keychain *C* level: SECRET | app_get_key(tmp)<br>A.k = tsm_kchain_add(C, tmp) |
| 6: A high integrity user at a low confidentiality level wants to append low confidentiality, high integrity keys to a high confidentiality, high integrity keychain. | SECRET/ HIGH_INTEG | Key *K* level: SECRET/HIGH_INTEG<br>Keychain *C* level : TS/HIGH_INTEG | Application operations: ss_kchain_append(A.c, A.k); |

**Note that Scenario 4 does not address integrity of the keys and keychain. For example, a low integrity SECRET key could corrupt the TS keychain, and a low integrity application could flood SS with requests, leading to denial of service for other processes needing to use SS, or exhaustion of the TS keychain space. SC can be configured to avoid these problems through the use of integrity labels, which would prevent writing low integrity data to a high integrity object. See scenario 6.*

# 4 Security Analysis

In this section we provide security analysis of the hardware and the integrated software architecture and show how the strong confidentiality and integrity properties of the SP architecture are preserved after virtualization and integration with the TML. In the first part we show that the new hardware instruction introduced do not compromise the confidentiality of user secrets. In the second part, we show that the assurance of the TML is not reduced due to the integration and the assurance of enforcement of MAC policies on the user's secrets as as high as the assurance of enforcement of MAC policies by the TML.

## *4.1 Security analysis of hardware*

The goal of the original SP architecture [rlee05a] is to protect the syntactic integrity and confidentiality of user keys. This architecture aims to protect the keys from some simple hardware attacks (probing memory bus, reading or modifying external memory or disk storage ) and software attacks that could be mounted using a compromised OS. Threats from probing internal registers inside the processor and physical side channel attacks like differential power or timing analysis are not considered. Also, only operational threats and not developmental threats are considered. In this analysis section, we show that the virtualized SP performs as well as the original SP design under this threat model.

Virtualization of the CEM state of an SP processor requires the TML to save and restore the three CEM registers on any VM switch. The Save SPregs and Restore SPregs instructions are only available to the SK in ring -2, which is expected to save to memory space inaccessible from any other ring. Therefore, virtualization does not expose the CEM state to any software outside of ring -2 or allow modification by any untrusted software. The instructions save and restore the CEM registers in one atomic operation. This ensures that the CEM state in the processor remains consistent, either in Interrupted_CEM mode or Normal mode. The virtualization instructions are not permitted in Active CEM mode so restoring will never resume Active_CEM state directly. Rather, the native SP protection where a return to Active_CEM state is only allowed when the SP hardware detects a jump to the correct saved CEM interrupt return address with the correct saved CEM interrupt hash value for the encrypted general registers.

As trusted software, the SK is assumed to save and restore the CEM state properly. It should protect the confidentiality and integrity of the saved state in memory, always restore the correct state for each block, and not

leak data from one block to another. The SK also properly initiates the CEM state for a new block. Despite this assumption, we describe the potential attacks on each of the CEM registers. The CEM.IntHash prevents previously valid CEM sessions from being replayed and detects modifications to the general registers. The SK cannot generate a valid keyed hash to match arbitrary modifications to the register ciphertext. However, any previously saved register state will carry a valid hash and could be replayed. By colluding with the OS, an attacker could try to use the old encrypted registers, the SK could restore the corresponding stale hash and allow CEM to reenter with old values. The CEM.IntAddr allows CEM to be resumed without assistance from the OS. It also ensures that CEM only resumes at the same place it was interrupted. The SK could change this address and resume CEM at a different point in the TSM, however, if it tries to resume to non-TSM code, the code integrity checking will fail on the first instruction, preventing illegitimate access to CEM data. Finally, SP.status keeps track of the current processor mode. "Active CEM" is not valid during a restore and will not be accepted. Incorrectly assigning "Interrupted CEM" or "Normal" modes will cause an exception when resuming CEM or cause an interrupted CEM state to be ignored, respectively. Most of these attacks simply result in denial of service, and a lost CEM task can be restarted. Thus the SK will not be able to compromise the confidentiality of keys stored in SP using any of the above described attacks.

Since the SK ensures strict partitioning between the states of the different blocks, we refer to the security analysis of SP in [rlee05a] to show that the design is immune to replay attacks within a given block. Further, the TML ensures isolation between the blocks and hence the intermediate CEM data from one block which could include user's secrets like cryptographic keys encrypted by the DMK, cannot be replayed within the context of another block. This prevents the possibility of a user's secrets like cryptographic keys from one security/integrity level from being transferred, corrupted or used at a different security/integrity level by replay attacks. Even if the MAC policies permit the flow of information from one security/integrity level to another, TML will ensure that this occurs via a chain of legal interfaces thus preventing direct inter-block replay attacks. Thus virtualization does not affect the confidentiality of user's secrets stored using SP.

## *4.2 Security analysis of integrated hardware/software architecture*

As referenced below, programs also exhibit both syntactic and semantic integrity.[2] In the first case, the integrity metric is the program's original value (e.g., manufactured, installed, or shipped image); for semantic integrity, the metric is the program's intended behavior (e.g., computer security policy, documented functionality or design documentation), including the notion that the code does not provide functionality beyond that which was intended (e.g., contain hidden behavioral artifacts) [irvi01]. With in mind, we present the security analysis of the virtualization and integration effort in two parts, each covering the relevant aspects of the architecture.

### 4.2.1 Assurance of enforcement of MAC on User's Secrets

The TML enforces MAC on resources that are exported by the SK. We use this to ensure that MAC is enforced on user's secrets like key chains and consequently on the use of keys to perform encryption and decryption operations. The key chains are managed as an MLS file system exported by the SK. The MAC policies are enforced at the granularity of keys based on the key's security/integrity level. Since the MLS file system manages its metadata explicitly as labeled kernel objects, it cannot be used to covertly signal across partitions. Since the MAC policies are enforced by the SK, the key chain manager/TSM or any other piece of software outside the SK that handle cryptographic keys need not be MLS aware. This not only simplifies the design and assurance of the application software but also increases the assurance of enforcement of MAC policies on cryptographic keys.

### 4.2.2 Assurance of TML

The TML is designed to enforce an MLS security policy regarding access to and flow of information with respect to the resources that it controls. TML controls all hardware resources, and from them exports certain abstractions (such as blocks, subjects, synchronization objects, devices and memory segments). TML is designed to protect

[2] Various policies are applied to programs with respect to integrity, such as the Biba policy [biba77], and "program integrity" [shir81], which refers to restrictions that would allow a program to execute, but not read, programs of greater integrity.

controlled resources from software and network-based attacks, as well as from physical attacks outside (e.g., observation of modification of data in transit between the processor and memory or disk) the processor boundary through the use of SP In general, the kernel enforces a lattice flow policy among blocks, and the SS virtualizes some of the kernel's exported resources and provides human-readable sensitivity labels for the blocks.

The SC architecture depends on the identified hardware protection mechanisms to ensure that subjects can neither read, execute nor write to the TML, except through interfaces that it controls. The hardware is configured so that privileged instructions that effect system security, such as those for establishing or changing a program's privilege level, are only available to the VMM privilege domain. The system is initialized so that the TML is the only software that runs in VMM privilege domain, and it assigns no program to the VMM privilege domain, so it alone can execute in the VMM privilege domain. As a result, the TML is tamperproof (i.e., syntactically integral) to the level of its *semantic* integrity. The TML is built to high assurance standards to ensure high semantic integrity .

The structure of the TML itself significantly contributes to its semantic integrity as well as its ability to protect itself [ncsc85]. The TML is internally structured into well-defined largely independent modules. It uses hardware segmentation to separate those elements that are protection-critical from those that are not. The TML modules are designed according to the principle of least privilege so that a failure in one component is isolated.

The TML incorporates significant use of layering, abstraction and data hiding. The design also excludes modules that are not protection-critical, which supports the design goal of simplicity.

SecureCore architecture ensures that the TML mediates every reference to its controlled resources. TML exports memory and IO *descriptors,* which the hardware ensures are the only way to access memory and the device resources. The TML ensures that these descriptors represent data movement that conforms to its policy – after descriptors are issued to subjects, their correct use is ensured by the hardware (i.e., a "read" descriptor cannot be used for modification). Encapsulated objects (e.g., for process synchronization) are checked by the TML for policy conformance on every reference. TML does not support direct subject-to-subject communication.

# 5 Related Work

Hardware-assisted virtualization is becoming ubiquitous. Intel and AMD have introduced CPU extensions for secure virtualization of shared resources [negi06, inte06, adva06]. The recently announced Trusted Execution Technology (TET) [intel], formerly known as LaGrande Technology, further enhances platform security by incorporating additional hardware security enhancements, including the Trusted Platform Module (TPM) [tcgp06] to protect secrets stored on the platform and to cryptographically "measure" software [intc06]. One of the new features in TPM V1.2 is the ability to differentiate commands issued from different sources, i.e., locality. This capability affords the existence of multiple simultaneous roots-of-trust on the platform. The Safer Mode Extension (SMX) in TET depends on the TPM locality support to establish a protected environment within the platform. The TET domain manager (running in the protected environment) is responsible for managing the TPM such that domain-specific secrets are not accessible across domains. The TET virtualization approach differs from our approach to virtualize SP in several ways. Firstly, unlike TET, SP is minimalist and is highly integrated with the ISA of the processor. The boundary of trust of TET covers a much larger set of components outside the processor. Due to the simplicity of SP, the architecture can subjected to rigorous analysis and hence has the potential for higher assurance. Secondly, TET depends on TPM for storage of user secrets and hence the user secrets are not as portable as it is in the SP architecture. Thirdly, unlike SP, TET does not protect the user's secrets cryptographically in memory. All these aspects are critical in the case of mobile devices in a high assurance environment. Further, our method focuses on virtualizing shared resources for MLS use, i.e., allowing multiple security domains to access CEM without covert channels, and providing an MLS file system service for organizing SP-related data.

Virtualizing devices typically requires a special software component that provides the abstract device interface and the appearance of multiple devices. Berger et al. describe an approach to implementing this component as a dedicated virtual machine running on a hypervisor [berg06]. In contrast to this approach, a traditional security kernel would implement this component as a kernel device driver [ncsc95]. Regardless of the design, a proper implementation of this component must ensure that no state persists across different instantiation of the device.

In short, our work differs most from other related work by the simultaneous use of

- highly integrated processor based crypto protection mechanisms in an MLS environment to obtain strong confidentiality protection of user's secrets like cryptographic keys not only during storage, but also during its use.
- high assurance separation kernel to perform access control on user secrets

# 6 Conclusion and Future Work

## *6.1 Conclusion*

This paper presented an approach to integrating processor-basedcrypto services in a separation kernel architecture. The integration provides strong cryptographic protection of the kernel and user controlled protections of secrets. These results can be used to protect cryptographic keys as we demonstrated in our reference examples. The integration also provides the foundation TML to enforce MLS polices on key chains as well as the interfaces of TSMs. These results are important in that they provide the foundation for secure mobile hand held devices such as those found in DoD future combat systems and civilian first responder applications. Specific important contributions of this work are

- a reference design for the virtualization and integration of processor-based cryptographic key management in a separation kernel system architecture such that
  - → the strong confidentiality (of user's secrets) guarantees provided by SP is preserved
  - → the assurance of enforcement of MAC policy is not reduced
- Elegant, simple design of SP Virtualization architectural extensions (only 2 new atomic virtualization instructions, transparent setting of CEM state between block switches, and mapping SK block switching to other SP interrupts to preserve SP protection).
  - → Does not introduce covert channels between blocks and sensitivity levels.

Through the ongoing SecureCore project, the design described in this paper is being captured in a worked example that validates design decisions and security analyses.

The Secure Core architecture provides a secure environment for SP features in which fine grained access control over interfaces of TSM may be achieved. The following section describes this and other future work in detail.

## *6.2 Future Work*

As a part of future work we hope to address the following

1. Multiple UMKs
   a) Currently the UMK is not virtualized and a single UMK is entered by the user and is shared between VMs. If future scenarios warrant virtualizing the UMK, additional SP hardware mechanisms and their integration with SK will be needed.
   b) Potential features:
      - Allow the user to enter multiple UMKs simultaneously
      - Control which software is authorized to use a particular UMK.
2. Configuration of SP
   a) Enable SP to be configured during initialization to allocate specific SP instructions to specific privilege levels. To be most general, e.g., supporting architectures other than SK, it will be useful for SP to be managed from either Ring -1 or Ring -2, and on processors without VMM privilege domain, from a trusted OS module in Ring 0 (e.g., for trusted scheduling to prevent DOS attacks.)
3. Additional access control on the use of CEM instructions
   a) Over and above the privilege level requirements for accessing SP instructions (see Table 2) individual system and application designs may require further restrictions on which processes may use the

*application* SP instructions.

## Acknowledgments

We acknowledge the support of our sponsors: the National Science Foundation and the Defense Advanced Projects Research Agency. This work has been supported under grants CNS-0430566, CNS-0430487, and CNS-0430598. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation or of DARPA.

## References

[adva05] Advanced Micro Devices, AMD "Pacifica" Virtualization Technology, 2005.

[adva06] Advanced Micro Devices, "AMD64 Architecture Programmer's Manual Volume 2: System Programming," Publication No. 24593, Revision 3.12, September 2006.

[barh03] Barham P, et. al., "Xen and the art of virtualization", Proceedings of the nineteenth ACM symposium on Operating systems principles, October 19-22, 2003, Bolton Landing, NY, USA

[benz05] Benzel, T. V., Irvine, C. E., Levin, T. E., Bhaskara, G., Nguyen, T. D., and Clark, P. C., "Design Principles for Security", NPS-CS-05-010, Naval Postgraduate School, September 2005.

[berg06] Stefan Berger, Ramon Caceres, Kenneth A. Goldman, Ronald Perez, Reiner Sailer, Leendert Van Doorn, "vTPM: Virtualizing the Trusted Platform Module," IBM Technical Report, RC23879 (W0602-126) February 14, 2006

[Biba77] K. J. Biba, "Integrity Considerations for Secure Computer Systems", MTR-3153, The MITRE Corporation, June 1975; ESD-TR-76-372, April 1977.

[devi98] Devine S, Bugnion E, and Rosenblum M., "Virtualization system including a virtual machine monitor for a computer with a segmented architecture", Technical Report US Patent 6397242, VMware, Oct 1998.

[gold72] Goldberg, R., "Architectural Principles for Virtual Computer Systems" . Ph.D. thesis, Harvard University, Cambridge, MA, 1972.

[intc06] Intel Corp., "LaGrande Technology, Preliminary Architecture Specification," September 2006.

[inte05] Intel Corp, "Intel® Virtualization Technology Specification for the IA-32 Intel® Architecture", 2005.

[inte06] Intel Corp., "IA-32 Intel Architecture Software Developer's Manual, Volume 3A: System Programming Guide, Part 1," June 2006.

[intel] Intel Corp., "Intel Trusted Execution Technology Architecture Overview," at http://www.intel.com/technology/security

[irvi01] C. Irvine and T. Levin, "Data Integrity Limitations in Highly Security Systems", Proc. of 2nd Annual Systems Security Engrg. Conf., Orlando, FL,March 2001

[irvi05] Irvine, C. E., Benzel, T, et. al. , National Science Foundation Poster - CyberTrust meeting, Sept 26, 2005

[irvi95] Cynthia Irvine,. "A Multilevel File System for High Assurance," Proceedings 1995 IEEE Symposium on Security and Privacy, Oakland, CA, pp. 78-87, May 1995.

[ncsc95] National Computer Security Center, "Final Evaluation Report Gemini Computers, Incorporated Gemini Trusted Network Processor Version 1.01," Report No. 34-94, 28 June 1995

[nscs85] National Computer Security Center, "Trusted Computer System Evaluation Criteria",(orange book), Department of Defense, No DOD 5200.28.STD, 1985.

[negi06] Gil Negier, Amy Santoni, Felix Leung, Dion Rogers, Rich Uhlig, "Intel Virtualization Technology: Hardware Support for Efficient Processor Virtualization," Intel Technology Journal, Vol. 10, Issue 03, August 2006.

[neum04] Peter G. Neumann. “Principled assuredly trustworthy composable architectures”, Technical Report CDRL A001 Final Report December 28, 2004, SRI, Menlo Park, CA, 2004.

[reed79] Reed, D.P., and R. K. Kanodia, “Synchronization with Event counts and Sequencers”, Communications of the ACM, 22(2):115-- 123, Februrary 1979.

[rlee05a] Lee, R., Kwan, P., McGregor, J.Dowskin, J., and Wang, Z., “Architecture for protecting critical secrets in microprocessors,” in Proc 32nd International Symposium on Computer Architecture, June 2005.

[rush82] John Rushby. Proof of Separability: A Verification Technique for a Class of Security Kernels. Proceedings of the 5th International Symposium on Programming, pages 352-62, 1982.

[salt75] Jerome H. Saltzer and Michael D. Schroeder. “The protection of information in computer systems”,. Proceedings of the IEEE, 63(9):1278–1308, 1975.

[shir81] Shirley, L.J., and R. Schell, "Mechanism Sufficiency Validation by Assignment," Proc. IEEE Symp. Security and Privacy, Apr. 1981, pp. 26-32.

[tcgp05] Trusted Computing Group, TPM Main, Part 1, “Design Principles, Specification Version 1.2, Revision 85”, 13 February 2005.

[tcgp06] Trusted Computing Group, “TPM Main Part 1 Design Principles,” Specification Version 1.2, Revision 95, 29 March 2006.

[vanf05] W. M. Vanfleet, R. W. Beckwith, B. Calloni, J. A. Luke, C. Taylor, and G. Uchenick, “Mils: Architecture for high assurance embedded computing,” CrossTalk, vol. 18, pp. 12–16, August 2005.