Intel® Software Guard Extensions (Intel® SGX) Support for Dynamic Memory Management Inside an Enclave

Frank McKeen, Ilya Alexandrovich, Ittai Anati, Dror Caspi, Simon Johnson, Rebekah Leslie-Hurd, Carlos Rozas

Intel Corporation

{frank.mckeen, ilya.alexandrovich, ittai.anati, dror.caspi, simon.p.johnson, rebekah.leslie-hurd, carlos.v.rozas}@intel.com

ABSTRACT

We introduce Intel® Software Guard Extensions (Intel® SGX) SGX2 which extends the SGX instruction set to include dynamic memory management support for enclaves. Intel® SGX is a subset of the Intel Architecture Instruction Set [1]. SGX1 allows an application developer to build a trusted environment and execute inside that space. However SGX1 imposes limitations regarding memory commitment and reuse of enclave memory. The software developer is required to allocate all memory at enclave instantiation. This paper describes new instructions and programming models to extend support for dynamic memory management inside an enclave.

1 INTRODUCTION

Software Guard Extensions (SGX) [1] [2] [3] [4] provides the capability to protect specified areas of an application from outside access. The area is called an enclave and hardware provides confidentiality and integrity for the specified area. SGX allows software developers to build trusted modules inside an application to protect secrets. A software developer specifies the contents of an enclave and a relying party can confirm that the area is instantiated correctly on a remote machine. To enforce isolation, hardware performs extra memory checks such that only enclave code can access enclave data. When the external software attempts to access the enclave, hardware will abort the accesses. In this manner SGX provides a trusted place to stand for application developers. The first release of the SGX architecture is referred to as SGX1.

While working with software developers, three shortcomings with the SGX1 were identified. First all enclave memory must be committed at enclave build time. This increases the build time. Committing memory places pressure on the enclave page cache (EPC), which is where enclave pages reside in memory and which is a limited resource. Lastly, the enclave cannot be easily adapted to a particular workload or environment. For example, an enclave developer must allocate memory for worst-case memory consumption of any workload. Otherwise, the enclave developer will need to release enclaves designed for different size workloads, each with a unique measurement. Likewise, the enclave developer must allocate memory for the maximum number of threads that the application may use for workload and platform combinations.

The second shortcoming is related to the management of access permissions associated with an enclave page. SGX extends the access permission model by associating an additional set of access permissions with enclave page that are stored in a SGX structure called the Enclave Page Cache Map (EPCM). The SGX enclave page permissions in EPCM are consulted after the page table's permissions checks have been applied. In SGX1, the permissions are associated when a page is added to an enclave and cannot be changed afterwards. This means that permissions need to be set to allow all possible usages of the page. For example, in order to relocate code inside an enclave, the access type must be left as Read, Write, Execute (RWX). The inability to modulate page permissions can constrain or limit certain capabilities inside an enclave such as garbage collectors, which require the enclave memory manager to periodically restrict write access to the page.

The last shortcoming is related to library OS support where secure exceptional handling and lazy loading code inside an enclave are important features. SGX1 has support for exception handling inside the enclave but deficiencies were identified of information recorded when a general protection fault or page fault occurs inside an enclave. Lazy loading of code is important to library OS usage as the entire application and Library OS runs inside the enclave which can be large. Deferring the loading of code presents unique security and functional requirements as compared to deferred commitment of heap and stack.

To address these problems six new instructions and new exception behavior were added to the SGX architecture known as SGX2. These instructions provide software with more capability to manage memory and page protections from inside an enclave. The new exception behavior provides page management capability. This paper describes theses instructions and explains how to use them.

This paper is divided into several sections. In Section 2, we provide an overview of the SGX security requirements and dynamic memory management requirements. Section 3 describes software flows to make use of the SGX2 instructions. Section 4 describes the SGX2 instruction set and software model. Section 5 describes some of the implementation and validation issues and concerns. It also describes how we addressed the concerns when implementing these instructions. Finally in section 6 we draw conclusions.

HASP '16, June 18 2016, ,

ACM 978-1-4503-4769-3/16/06…$15.00
DOI: http://dx.doi.org/10.1145/2948618.2954331

2 SGX2 Considerations & Requirements

An explicit goal of SGX2 was to address the shortcomings of SGX1 while maintaining the security properties of the SGX architecture and system software. To provide dynamic memory management several conditions need to be satisfied.

- Manipulating memory and permissions of an enclave must be done with the knowledge and consent of the enclave. This enables the enclave to manage its own security
- If enclave code is changed incorrectly or without knowledge of the enclave, execution should be suspended until the condition is resolved.
- The system resource manager (OS or VMM) must be able to manage and allocate the resources as requested using standard techniques and priorities.
- Manipulation of memory permissions involves both the system permissions and the EPCM permissions. EPCM permissions allow the enclave developer to specify the restrictions and access control for the enclave.

To achieve these requirements, the memory management function must be split between a system memory resource manager (system manager) which manages the system resources and an internal enclave resource manager (internal manager) which manages the enclave memory from inside the enclave. A protocol which consists of communication between the system manager and an internal manager is needed. This is shown in Figure 2-1.

The system memory manager is responsible for allocating memory, paging memory, changing permissions, and changing page types. This includes managing the page table entry permissions and initiating EPCM permissions of the enclaves. Changes to the EPCM permissions and page types are done using the instructions described in section 3. The system manager is also responsible for managing the TLBs of each thread.

The internal manager is responsible for starting memory change requests and verifying that the system manager has processed the requests correctly. In addition the internal manager can make some changes in EPCM permissions of an enclave page. The internal manager does not have direct access to the page tables and must request the system manager to make changes in page table entry (PTE) permissions. New instructions provide the internal manager the capability to check and commit the results of the requested operation.

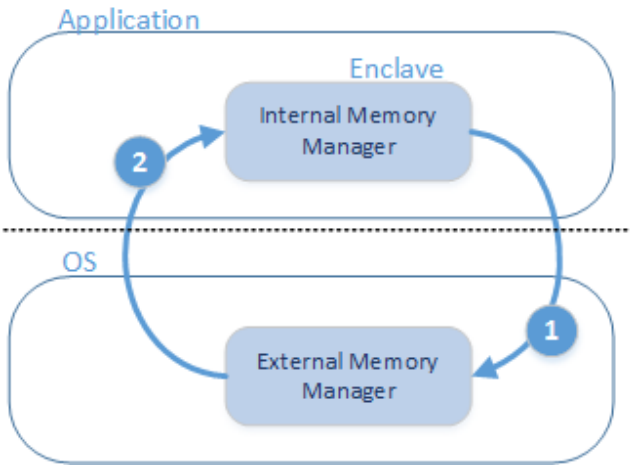

Figure 2-1 System and Internal Memory Managers

This division of work is shown in Figure 2-1. Step 1: the internal manager sends a request to the system manager. The system manager performs the requested operation(s). It then returns to the internal memory manager with a response indicating success or failure as shown in step 2. The internal memory manager then checks the result to ensure the request was fulfilled correctly. If the operation was performed correctly the results are committed. If there is an issue with the request, an exception or failure is generated. Details of the protocol and software usage are discussed in section 3.

2.1 Security Considerations

The software inside an enclave must be able to ensure that changes in permission do not affect the security of the enclave in a negative way. The hardware must protect enclave software from attacks as described in [5] and [6]. With dynamic memory management some new constraints and requirements are introduced.

Enclave software may change the permissions of a page to reduce the access capabilities or increase the access capabilities. For example software may make the page read only from read, write, executable. In this case the permissions are reduced. An example of increasing permissions would be setting a page from read to read, write, execute. When enclave software wishes to increase access permissions to a page it is acceptable for processors to switch to the more lenient permission lazily. The system manager can update the page tables and EPCM entries. If the translation lookaside buffer (TLB) contains the older, more restrictive permissions for an enclave page, a page fault occurs on the access. The act of exiting the enclave clears the TLB of the previously cached and the more restrictive address translations and permissions in the TLB. The system manager checks that the page is mapped correctly. It will return to the program and resume execution. The enclave resumes execution and subsequent access will load the TLB with the new and more lenient permissions. The enclave security is not compromised if the page is accessed with the older permissions.

When enclave software wishes to restrict page permissions, there must be a point where the enclave can be assured the permission restrictions are complete and the previous cached address translations or cached permissions are removed. SGX2 extends the SGX1 TLB checking mechanism used for enclave paging to guarantee to the enclave that the old permissions are removed from the TLBs. The details are discussed in section 3.3

Software applications which handle their own memory paging and exceptions need secure reporting of these exceptions. SGX2 extends SGX1's MISC field of the State Save Area (SSA) with the EXINFO field that holds information about memory paging faults and general protection faults

SGX1 allows the system memory manager to remove pages from an enclave using the EREMOVE leaf function. However, since the enclave doesn't participate in this process it doesn't know if the page has actually been removed. If an enclave wants to return pages to the system manager but be able to reuse the same addresses later, it needs to be told explicitly when the pages have been removed from the enclave. A 2-step protocol will be used to perform the operations. This protocol allows the enclave to know that the pages are removed. It can then reassign them as needed. Without the protocol an attacker could randomly substitute pages without the enclave's knowledge and permission. See section 3.2 for more information.

2.2 Software Considerations

The internal memory manager has certain functions it must perform to reallocate the memory resources. For example, when an

application needs to add a thread, pages must be allocated as Thread Control Structure (TCS), State Save Area (SSA) pages, and thread local variables. If the internal memory manager is allocating from a general pool of PT_REG pages, it must request the OS to change the page type to PT_TCS for TCS pages. In addition, it may want to remove execute capability from SSA pages. In order to change the page type the enclave memory manager requests that the system manager change the page type using one of the SGX2 instructions. If the enclave memory manager needs more recourses from the system manager it should call the system manager to allocate resources to the enclave using one of the SGX2 instructions

Exception Reporting Inside an Enclave

Usages, such as a library OS, receive exceptions inside an enclave. This allows the enclave software to inspect exceptions being reported to make sure the external software is providing the correct information. In this case the exception condition should be reported inside the enclave. The enclave can then look at the information as needed. SGX2 adds several exception conditions to the SSA frame when exiting an enclave. They include page faults (#PF) and general protection violations (#GP). Enclave developers can opt-in to receiving this additional information to allow the internal manager to approve the requests.

Demand Loading of Library Pages

Library pages are loaded on demand when the program touches the page. When page fault occurs software would like to load the page into the location of the fault. The internal manager must have a mechanism to load the page without allowing access until the copy is complete. SGX2 adds a leaf function to perform the copy securely. See section 3.5 for detailed steps.

3 SGX2 Overview and Usage

SGX1 enclave memory managers can support dynamic allocation of memory using malloc and free. However, they require enclave memory (heap) to be committed by the operating system at enclave build time. Furthermore, enclave memory managers have limited or no support for page permission modification, dynamic thread creation, and lazy loading of code. In addition, new software models related to library OSes require improved exception reporting.

Table 3-1 SGX2 Instruction Leafs

Leaf function	Description
ENCLS[EAUG]	Add a regular read/write accessible page of zeros to an already initialized enclave
ENCLS[EMODT]	Modify the type of an existing EPC page
ENCLS[EMODPR]	Restricts access permissions of an existing EPC page
ENCLU[EACCEPT]	Accepts page type modifications to the running enclave
ENCLU [EACCEPTCOPY]	Copies existing EPC page content into EAUGed page and accepts the page into the running enclave
ENCLU[EMODPE]	Extends access permissions of an existing EPC page

To improve the enclave memory management and library OS support, the SGX2 extension adds three enclave ENCLU leaf functions and three privileged ENCLS leaf functions. Table 3-1 provides a short description of the new leaf functions. The details of the new SGX2 leaf functions are discussed in section 4.

In addition to the new leaf functions, a new SGX page type, PT_TRIM, is defined to allow an enclave to remove a page from the enclave and reclaim the linear address for future use. Lastly, information about enclave generated General Protection Faults (#GP) and Page Faults (#PF) are store in the enclave's State Save Area (SSA).

A high level conceptual software protocol for using the new SGX2 leaf functions is illustrated in Figure 3-1 In the protocol, the enclave's internal memory manager starts by making a system call to the system manager requesting some operation (e.g. permission modification using EMODPR). The system manager will perform the operation and return back to the internal manager which then verifies and/or completes the operation (e.g. invokes EACCEPT). It is also possible to initiate the protocol using faults generated by the enclave. For example, enclave memory commitment can be driven by an enclave accessing an allocated but not yet committed memory.

3.1 Enclave Malloc

To implement dynamic memory allocation and commitment, the enclave runtime system and the operating system have to agree on a protocol that coordinates the operating system usage of EAUG and the enclave runtime system usage of EACCEPT.

The following is an example protocol:

1. Internal manager requests memory which results in the enclave runtime system to allocate memory from its internal pool of memory. When the memory pool runs low the internal manager requests the system manager to allocate more memory from the virtual address space.
2. The system manager allocates virtual address space but does not commit memory and returns a reference to the virtual address space to the internal manager
3. The enclave internal manager returns a reference to the enclave. When the enclave accesses the newly allocated memory, a page fault is generated as memory has not been committed.
4. The OS page fault handler detects that the virtual address has been allocated but memory has not been committed. The OS commits memory by using EAUG and maps the committed but pending page into the enclave address space. The OS then sends a signal to the enclave internal manager.
5. The internal manager receives the signal from the OS that memory has been committed for a particular address. The internal manager checks that the virtual address has been allocated but not yet committed. If those checks pass, the internal manager executes EACCEPT which allows the enclave to access the pending page. The signal handler returns back to the application which eventually results in the enclave execution resuming.

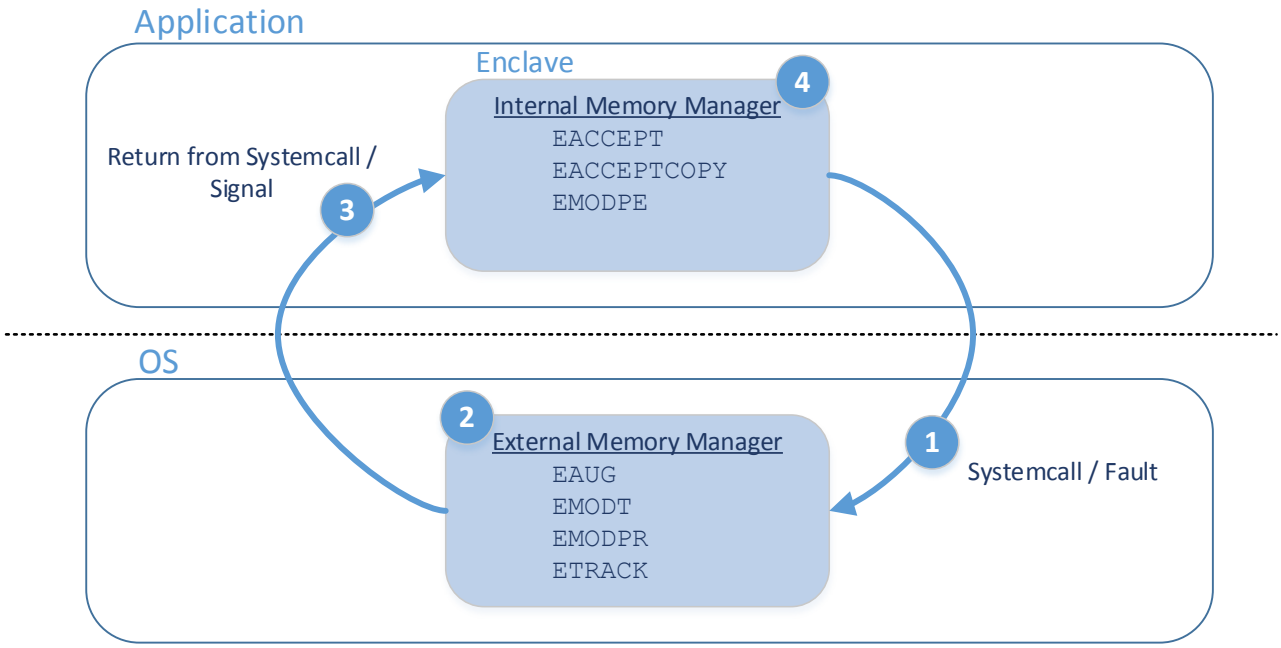

Figure 3-1 SGX2 Software Flow

3.2 Enclave Free

SGX1 allows for the virtual address space to be released and committed memory to be reclaimed with certain limitations. SGX2 provides a more robust reclamation of committed memory. EMODT is used to modify the EPCM of the page to prepare it for removal. As mappings to the page may be cached in a TLB, SGX2 extends ETRACK to track the flushing of TLBs when restrict accessing pages with EMODT and EMODPR. EACCEPT will verify that TLB mappings have been removed. Once again the OS and enclave runtime system need to coordinate the execution of EMODT, ETRACK, and EREMOVE by the OS and execution of EACCEPT by the enclave runtime.

The following is an example protocol:

1. The enclave releases allocated memory and the internal manager decides to release address space back to the OS.
2. The system manager executes EMODT on all pages being freed to change the page type to PT_TRIM and to clear the EPCM access permission bits. This begins the process of decommitting memory. The system manager then executes ETRACK on the SECS of the calling enclave and then sends IPIs to logical processors which may contain TLB mappings to the pages that had been trimmed.
3. Once all logical processors responded to the IPI, control is returned to the internal manager.
4. The internal manager verifies that committed memory has been decommitted by executing EACCEPT to verify that the pages have been trimmed using EMODT and all stale TLB mappings have been flushed. The internal manager needs to update its tracking information that the virtual address has no committed memory.
5. The system manager can later reclaim the committed memory by executing EREMOVE on the trimmed pages.

3.3 Changing Page Permissions

Enclave runtimes have varying reasons to change access permissions to a page. For example, an enclave loader may need to perform address relocations to pages which at runtime should not be writable. Thus before allowing execution the enclave loader needs to update the contents of the pages and then remove write permissions to the pages. The enclave runtime may have garbage collector which needs to periodically mark pages as read-only and then restore write permissions to the pages.

SGX2 has two separate flows for changing EPC page permissions. If the change is entirely permissive then the following protocol can be used:

1. The internal manager executes EMODPE to extend the page permissions in the EPCM.
2. The internal manager requests the system manager to extend page permissions in the page tables.

The order of operations can also be reversed if desired.

If the change in permission is purely restrictive, a more complex protocol is required as EMODPR will modify EPCM permissions to be more restrictive but TLB mappings with more permissive accesses may exist in some processors. The following is an example protocol:

1. The internal manager requests that the system manager to restrict permissions on a page.
2. The system manager executes EMODPR and updates page table permissions. After permissions have been updated, the system manager executes ETRACK on the SECS of the calling enclave and sends IPIs to all processors that may be executing inside the enclave to flush TLB mappings.
3. After all IPIs have been acknowledged, control is returned to the internal manager. The internal manager verifies that page permissions have been restricted and TLB mappings flushed by

executing EACCEPT.

If a permission change is both restrictive and permissive, for example, a change from read-write to read-execute, the internal manager should request the permission restriction first and then permission extension to avoid undesirable intermediate states.

3.4 Thread Control Structure Allocation

SGX2 supports the allocation of Thread Control Structures (TCS) at runtime. The software protocol is very similar to page restriction except that instead of executing EMODPR the OS executes EMODT.

The internal manager starts by picking a 4K page in the enclave's linear address space as described in 3.1 and initializes the page with appropriate TCS values. The TCS configuration values include the size and location of the State Save Area, the enclave entry point for the TCS, and the FS/GS base values.

Once the internal manager has determined an acceptable address, the following protocol can be used:

1. Internal manager initializes the contents of a regular EPC page with appropriate TCS values. If the enclave memory has not been committed then internal manager will need to perform a request to allocate memory as described in section 3.1. After the contents of the page have been initialized, the internal manager requests that the system manager convert the page to a TCS.
2. The system manager executes EMODT to set the page type to PT_TCS and to clear the EPCM access permission bits. The page is also marked modified which prevents the page from being used as a TCS until the enclave accepts the TCS via EACCEPT. The system manager then executes ETRACK. The system manager sends IPIs to flush all old mappings to the page and returns control to the internal manager.
3. The internal manager executes EACCEPT on the modified TCS page. EACCEPT will verify that TLB mappings have been flushed and perform consistency checks on the TCS page before clearing the modified bit and making the page available to EENTER.

3.5 Dynamic Loading of Modules

To support dynamic loading of modules, SGX2 provides EACCEPTCOPY which allows the internal manager to atomically initialize the contents and permission of a page. Here is an example protocol for allowing the dynamic loading of a module a page at a time:

1. First, the internal manager indicates to the system manager that a virtual address space has been allocated but not committed (same as in 3.1).
2. When an enclave attempts to access a page in this virtual address, a page fault is generated and the system manager commits memory by executing EAUG and signals the internal manager.
3. The internal manager identifies the virtual address as belonging to a module page to be loaded. The system manager may load the contents of the page into regular memory or the enclave runtime system may need to request the content be loaded into regular memory.
4. The internal manager then copies the contents of the module into private enclave memory. The internal manager should verify the integrity of the contents and apply any required relocations. Finally, the internal manager copies the contents and initializes permissions using EACCEPTCOPY.

The use of EACCEPTCOPY for dynamic loading is required because of potential races with other threads in the enclave. For example, if the internal memory manager attempted first EACCEPT and then initialize the contents and permissions, a second thread in the enclave could read the contents of the page before the page has been completely initialized or could modify the contents of the page before permission restrictions have occurred.

3.6 Library OS Support

Library OSes provide a new type of container where an application is bundled with an OS runtime that executes in user level (ring 3) [7] A critical requirement for SGX enabled library OSes is to provide secure exception handling. In particular, the library OS exception handler must be able to securely determine that an exception was generated by an enclave, the type of exception (e.g. page fault), and information about the faulting condition.

SGX2's enhanced enclave exception handling provides the enclave runtime information about exceptions occurring inside the enclave. This secure reporting of exceptions allows enclave runtime systems to invoke application exception handlers without relying on the external exception handling system.

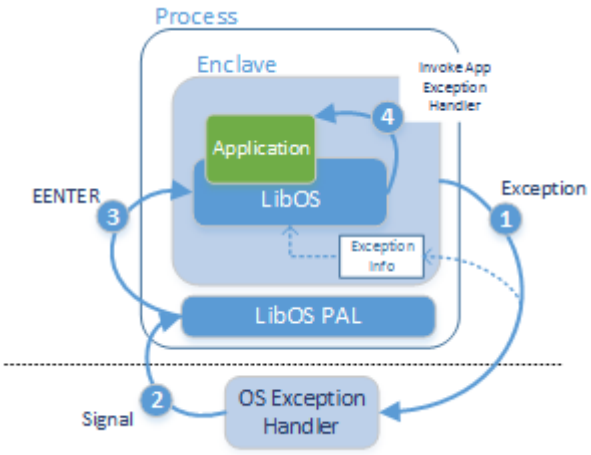

Figure 3-2 SGX enabled LibOS exception handling

Figure 3-2 illustrates the usage of enhanced exception handling in a SGX enabled LibOS.

1. The process begins with an exception generated inside an enclave. The processor records exception information in the Save State Area and delivers the exception to the OS exception handler.
2. If the OS cannot handle the exception, the OS signals the LibOS PAL (Platform Adaptation Layer) exception handler.
3. The LibOS PAL executes EENTER to invoke the LibOS exception handler inside the enclave.
4. The LibOS exception handler reads the exception information, generates an OS specific exception context, and invokes the application exception handler inside the SGX enabled LibOS.

4 SGX2 ISA

The SGX2 extension adds new SGX leaf functions and capabilities including three ENCLS leaf functions (EAUG, EMODT, and EMODPR) described in 4.1 and three ENCLU leaf functions (EACCEPT, EACCEPTCOPY, and EMODPR) described in 4.2. In addition to the new leaf functions, a new SGX page type, PT_TRIM, is used in conjunction with the new leaf functions. In addition to the new leafs, SGX1 leaf functions and structures have been enhanced to support SGX2 functionality. The ENCLS[ETRACK] leaf functionality has been enhanced to support tracking enclave page translations for SGX2 related activities and is described in 4.3. The

Security Information's (SECINFO) FLAGS field has been enhanced to include the new flag bits "Pending", "Modified", and "Permission Restriction". This information is saved by ENCLS[EWB] and restored by ENCLS[ELDU/ELDB].

Separate from the new leaf function, SGX2 allows recording information concerning General Protection Faults (#GP) and Page Faults (#PF) exceptions from within the enclave which is described in 4.4.

To help system memory managers to determine whether a page fault was generated by violation of access permissions of page tables versus violation of access permissions of the EPCM, a new Page Fault Error Code bit called PFEC.SGX was added and is described in 4.5.

4.1 ENCLS leaf functions

EAUG

This leaf function augments the enclave with a page of zeros by zeroing a page of EPC memory, associating that page with an SECS page, and updating the linear address and security attributes in the page's EPCM entry. A successful execution of the function puts the page in "Pending" state. The operation receives two input parameters, a pointer to the destination page in EPC, and a pointer to the enclave's SECS page. While in "Pending" state, the page cannot be accessed by anyone, including the enclave. Only after the enclave approves the page by using the ENCLU[EACCEPT] leaf function, will the page be accessible to the enclave.

EMODT

This leaf function modifies the type of an EPC page and puts the page in "Modified" state. Allowed page types are PT_TCS and PT_TRIM. The operation receives two input parameters, a pointer to the target page in EPC, and a pointer to the page's new security attributes. While in "Modified" state, the page cannot be accessed by anyone, including the enclave. Only after the enclave approves the page by using the ENCLU[EACCEPT] leaf function, will the page be accessible to the enclave.

EMODPR

This leaf function restricts the access rights associated with an EPC page of an initialized enclave and puts the page in "Permission Restriction" state. The operation receives two input parameters, a pointer to the target page in EPC, and a pointer to the page's new security attributes. The operation will fail if it attempts to extend the permissions of the page. While in "Permission Restriction" state, the page cannot be accessed by anyone, including the enclave. Only after the enclave approves the page by using the ENCLU[EACCEPT] leaf function, will the page be accessible to the enclave.

4.2 ENCLU leaf functions

EACCEPT

This leaf function must be executed from within an enclave. It accepts changes to a page in the running enclave by verifying that the security attributes specified in SECINFO match the page's security attributes in EPCM. The operation receives two input parameters, a pointer to the target page in EPC, and a pointer to the page's approved new security attributes. After a successful execution of EACCEPT the page's "Pending", "Modified", or "Permission Restriction" state is cleared and the page becomes accessible to the enclave.

EACCEPTCOPY

This leaf function must be executed from within an enclave. It copies the contents of an existing EPC page into an uninitialized EPC page that was created by EAUG. The operation receives three input parameters, a pointer to the target page in EPC, a pointer to the page's new security attributes, and a pointer to the page's new content. After a successful execution of EACCEPTCOPY the page's "Pending" state is cleared and the page becomes accessible for the enclave

EMODPE

This leaf function must be executed from within an enclave. It extends the access rights associated with an existing EPC page in the running enclave. The operation receives two input parameters, a pointer to the target page in EPC, and a pointer to the page's new security attributes. The operation will fail if it attempts to restrict permissions of the page. Since the execution happens from within the enclave, it's trusted and takes effect immediately.

4.3 Managing Page Table Translations

SGX2 extensions modify the access rights of pages of an initialized enclave. As such, any stale page table translation must not compromise the security objective of SGX. The ENCLS[ETRACK] leaf function used in SGX1 in the paging process is extended for SGX2 to ensure that stale page table mappings are removed from TLBs prior to committing the page's new attributes. Table 4-1 describes the TLB synchronization requirements for the SGX2 leaf functions.

EMODPE is a special case that doesn't require TLB synchronization for security, but software could use the synchronization mechanism to guarantee that all threads will observe the new permissions.

Table 4-1 Page Table Tracking

Leaf function	TLB synchronization
EAUG	Not required
EMODT	Required. Enforced by EACCEPT
EMODPE	Optional.
EMODPR	Required. Enforced by EACCEPT
EACCEPTCOPY	Not required

4.4 Enclave Exception Handling Enhancements

In the SGX architecture, whenever an exception or interrupt occurs while executing inside an enclave, the processor performs an asynchronous enclave exit (AEX) which securely saves the state of the processor inside the enclave's SSA, replaces it with synthetic values, and exits the enclave. In addition to saving the state of the processor, the cause of the AEX is stored in the EXITINFO field in the SSA. This allows an enclave exception handler to determine the cause of the exception and attempt to resolve the cause of the exception. SGX2 extends this mechanism and provides enclave writers an opt-in to save information about page fault and general protection fault exceptions inside the MISC field of the SSA by setting the EXINFO bit (bit 0) in the SECS.MISCSELECT field during enclave creation. If SECS.MISCSELECT.EXINFO bit is set, the processor saves #PF and #GP information into the EXINFO structure as shown in Table 4-2:

Table 4-2 EXINFO Field

Field	Offset (bytes)	Size (bytes)	Description
MADDR	0	8	If #PF: contains the page fault linear address that caused a page fault. If #GP: the field is cleared
ERRCD	8	4	Exception error code for either #GP or #PF

4.5 EPCM-induced Memory Fault Reporting

When a memory reference is unable to complete due to settings in the EPCM which prevent the access, a #PF exception is generated. A bit in the Page Fault Error Code (PFEC) indicates that the page fault was due to EPCM access checks. This bit is located at bit position 15 and called "SGX".

The SGX bit is set only if **there** is a valid translation for the linear address and its cumulative Paging/EPT access rights permit the access. The processor always provides SGX bit in the page fault error code pushed onto the stack of the exception handler. This bit is also provided inside the enclave in EXINFO if the MISCSELECT field has enabled it.

Processors that support SGX2 also support the SGX page fault error code without any explicit opt-in by system software. This bit is provided inside the enclave in EXINFO if the MISCSELECT field has enabled it.

5 SGX2 Validation

SGX maintains a large amount of global security sensitive data for its operation. This includes per-page metadata stored in the EPCM, per-enclave and per-thread control structures (SECS, TCS) and page versions (VA). Accesses to this data, which is shared across all logical processors, must be appropriately synchronized to maintain security, while still maximizing parallelism for performance. These sometimes contradicting requirements along with the need to be compatible with both system software and application operation, necessitated the usage of complex synchronization mechanisms in SGX.

Concurrency-related complexity in SGX2 is greatly increased over SGX1. Some previous restrictions on the software concurrent operation have been lifted. Page Table translation tracking now serves multiple, potentially concurrent, processes. In addition to paging, it is now used for freeing pages, adding new enclave threads and changing page permissions.

5.1 Formal Modeling and Verification

Applying formal verification techniques very early in the design process allowed us to find pernicious concurrency bugs and to increase our confidence that critical errors in the design were not being overlooked.

A formal model of SGX was created; its main goal was to detect bugs related to multi-threaded operation of SGX, which would impact its security properties. The model concentrated on the complex aspects of the architecture, where non-trivial synchronization mechanisms are used, especially around enclave page management and Page Table translations tracking.

Formal verification is commonly used at Intel for arithmetic and protocol validation. SGX instruction flows resemble concurrent software algorithms, where multiple threads access shared data structures. Linearizability for these type of algorithms is extremely helpful for validation. In a linearizable system, one cannot observe the difference between a sequentialized trace where each instruction executes atomically at its linearization point and a real trace that arises in the concurrently executing system. SGX is linearizable if each instruction appears to take effect atomically at some moment in time between its invocation and termination, called its linearization point [8]

An SGX-based system is complex to model as a whole. Several optimization techniques were applied in order to model SGX within reasonable run times:

- Management of an enclave page is (almost) independent of other pages. Thus, a single concise page is modeled.
- An enclave is independent of other enclaves. Thus, a single concise enclave is modeled.
- SGX control fields that have small or no relevance to security (as impacted by concurrent operation) are selectively ignored.
- The model uses a "worst case" approach, e.g., it assumes a Logical Processor creates a translation to an EPC page as soon as it enters enclave mode, while in reality this would only happen when the processor actually accessed the page.
- The initial state space is pre-calculated, saving the large number of modeling steps it would require to achieve all possible states.

Using formal verification, we have detected several concurrency-related bugs during SGX architecture development.

5.2 Implementation-Level Validation

For the same reasons mentioned above, much of the implementation-level validation effort was dedicated to multi-threaded validation.

Concurrency issues may be difficult to detect by examining the architectural state of the processor. Therefore, white-box checking was used, monitoring the micro-architectural state during SGX instruction execution. A generic data race detector [9] was applied, in addition to multiple dedicated checkers, many of which were derived from the formal model's assertions and verified similar linearizability criteria.

Many concurrency issues express themselves only under certain inter-thread timing conditions. To increase the chance of recreating such timing during validation, pseudo-random time delay "bubbles" were inserted in selected places in the SGX flows (e.g., near inter-thread synchronization points such as lock acquisition and release).

Several coverage criteria were applied. One criterion is synchronization coverage [10]. Other include a table of cross-instruction interaction and multi-instruction sequences.

6 Summary and related work

We have shown that adding new instructions to the current SGX1 programming model can provide better software development environment while maintaining the security properties of the enclave. The SGX2 instructions enable better protection of proprietary code which can be loaded and then protected using the EPCM. These instructions also allow for dynamic memory and threading support which is common among programming models. In addition features were added to support dynamic allocation of library pages in the library OS environment.

Using these instructions, developers can produce enclaves which can more robustly respond to individual system configurations and requirements.

7 ACKNOWLEDGEMENTS

The authors would like to thank the Microsoft Haven development

team, Andrew Baumann, Marcus Peinado, and Galen Hunt, They provided great feedback and insight as the instruction set was developed. [7] Their feedback and guidance provided valuable insight into the issues faced by advanced applications and library OS'. This groundbreaking work of porting a library OS into an enclave provided the impetus to provide secure page faulting inside an enclave as well as several other changes in the instruction set.
The authors of this paper wish to acknowledge the contributions of many hardware and software architects, designers, and validators who have worked in developing this technology.
Finally the authors would like to thank Krystof Zmudzinski and Meltem Ozsoy for their help in reviewing and critiquing this paper.

8 References

[1] Intel Corp, "http://www.intel.com/content/www/us/en/processors/architectures-software-developer-manuals.html," Intel, April 2016. [Online]. Available: http://download.intel.com/products/processor/manual/325462.pdf.

[2] F. McKeen, I. Alexandrovich, A. Berenzon, C. Rozas and H. Shafi, "Innovative Instructions and Software Model for Isolated Execution," in *HASP 2013*, Tel Aviv, Israel, 2013.

[3] I. Anati, S. Gueron, S. Johnson and V. Scarlata, "Innovative Technology for CPU Based Attestation and Sealing," in *HASP 2013*, Tel Aviv, Israel, 2013.

[4] M. Hoekstra, R. Lal, P. Pappachan, C. Rozas and V. Phegade, "Using Innovative Instructions to Create Trustworthy Solutions," in *HASP 2013*, Tel Aviv Israel, 2013.

[5] K. Brannock, P. Dewan, F. McKeen and U. Savagaonkar, "Providing a Safe Execution Environment," *Intel Technology Journal,* vol. 13, no. 2, 2009.

[6] V. Costan and S. Devadas, "Intel SGX Explained," https://eprint.iacr.org/2016/086.pdf, 2016.

[7] A. Baumann, M. Peinado and G. Hunt, "Shielding Applications from an Untrusted Cloud with Haven," in *11th USENIX Symposium on Operating Systems Design and Implementation*, Broomfield CO, 2014.

[8] R. Leslie-Hurd, D. Caspi and M. Fernandez, "Verifying Linearizability of Intel Software Guard Extensions," in *Proc. Of the 27th International Conference on Computer Aided Verification (CAV)*, 2015.

[9] X. T. W. T. Markus Metzger, "User-Guided Dynamic Data Race Detection," *International Journal of Parallel Programming,* vol. 43, no. 2, pp. 159-179, 2015.

[10] A. Bron, E. Farchi, Y. Magid, Y. Nir and S. Ur, "Applications of synchronization coverage," in *10th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP'05), USA, pp. 206-212, 2005*.

[11] D. Lie, M. Thekkath, M. Mitchell, P. Lincoln, D. Boneh, J. Mitchell and M. Horowitz, "Architectural Support for Copy and Tamper Resistant Software," in *Proc. of the 9th International Conference on Architectural Support for Programming Languages and Operating Systems*, 2000.

[12] G. Suh, D. Clarke, B. Gassend, M. van Dijk and S. Devadas, "AEGIS: Architecture for Tamper-Evident and Tamper-Resistant Processing," in *Proc. of the 17th International Conference on Supercomputing*, 2003.

[13] R. B. Lee, P. C. S. Kwan, J. P. McGregor, J. Dwoskin and Z. Wang, "Architecture for Protecting Critical Secrets in Microporcessors," in *Proc. of the 32nd annual International Symposium on Computer Architecture*, 2005.

[14] D. Champagne and R. Lee, "Scalable architectural support for trusted software," in *16th International Symposium on High Performance Computer Architecture (HPCA)*, 2010.

[15] J. Szefer and R. Lee, "Architectural Support for Hypervisor-Scure Virtualization," in *Proc. of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2012.

[16] Rick Boivie, IBM Corp, "Secure Blue++: CPU Support for Secure Execution," IBM Watson Research Center, Yorktown Heights NY, 2012.

[17] F. McKeen, U. Savagaonkar, C. Rozas, G. Michael, H. Herbert, A. Altmann, G. Graunke, D. Durham, S. Johnson, M. Kounavis, V. Scarlata, J. Cihula, S. Jeyasingh, B. Lint, G. Neiger, D. Rodgers, E. Brickell and J. LI, "METHOD AND APPARATUS TO PROVIDE SECURE APPLICATION EXECUTION". WPO Patent WIPO Patent Application WO/2010/057065, 14 November 2009.

"undefined". Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The information here is subject to change without notice. Do not finalize a design with this information.

No computer system can provide absolute security under all conditions. Built-in security features available on select Intel® processors may require additional software, hardware, services and/or an Internet connection. Results may vary depending upon configuration. Consult your system manufacturer for more details.
