Innovative Instructions and Software Model for Isolated Execution

Frank McKeen, Ilya Alexandrovich, Alex Berenzon, Carlos Rozas, Hisham Shafi,
Vedvyas Shanbhogue and Uday Savagaonkar
Intel Corporation

{frank.mckeen, ilya.alexandrovich, alex.berenzon, carlos.v.rozas, hisham.shafi,
vedvyas.shanbhogue, uday.savagaonkar }@intel.com

ABSTRACT

For years the PC community has struggled to provide secure solutions on open platforms. Intel has developed innovative new technology to enable SW developers to develop and deploy secure applications on open platforms. The technology enables applications to execute with confidentiality and integrity in the native OS environment. It does this by providing ISA extensions for generating hardware enforceable containers at a granularity determined by the developer. These containers while opaque to the operating system are managed by the OS. This paper analyzes the threats and attacks to applications. It then describes the ISA extension for generating a HW based container. Finally it describes the programming model of this container.

1 INTRODUCTION

Today's computer systems handle increasing amounts of important, sensitive, and valuable information. This information must be protected from tampering and theft. An entire industry is devoted to stealing information such as banking data or corporate intellectual property from systems [1]. There are many applications which must keep a secret on a platform. Some example applications are financial programs, ebanking, and medical records programs. Each secret holder may be mutually distrustful of other secret holders. Each secret must be protected independently of the other secrets. This paper describes Intel® Software Guard Extensions, (Intel® SGX), a set of new instructions and memory access changes added to the Intel® Architecture. These extensions allow an application to instantiate a protected container, referred to as an enclave. An enclave is a protected area in the application's address space, Figure 1, which provides confidentiality and integrity even in the presence of privileged malware. Attempted accesses to the enclave memory area from software not resident in the enclave are prevented even from privileged software such as virtual machine monitors, BIOS, or operating systems.

SGX allows the protected portion of an application to be distributed in the clear. Before the enclave is built the enclave code and data is free for inspection and analysis. The protected portion is loaded into an enclave where its code and data is measured. Once the application's code and data is loaded into an enclave, it is protected against all external software access. An application can prove its identity to a remote party and be securely provisioned with keys and credentials. The application can also request an enclave & platform specific key that it can use to protect keys and data that it wishes to store outside the enclave.

In addition to the security properties, the enclave environment offers scalability and performance associated with execution on the main CPU of an open platform.

Supporting SGX involves two major additions to the Intel Architecture. First is the change in enclave memory access semantics. The second is protection of the address mappings.

This paper is divided into several sections. In Section 2, we provide an overview of the SGX protection model. Section 3 describes the SGX instruction set and software model. Section 4 describes the hardware components used to support an enclave for an application. Section 5 describes the enclave creation process. Section 6 describes how an application transitions in and out of an enclave. Section 7 describes how enclaves can be paged out of the protected memory to allow for multiple or very large enclaves. Finally, in section 8, we summarize the benefits and show where this technology contains novel enhancements to advance security in open systems.

The SGX architecture also includes instructions and architecture for remote attestation and sealing. This is described in [2]. In addition some important usages developed at Intel Labs are described in [3].

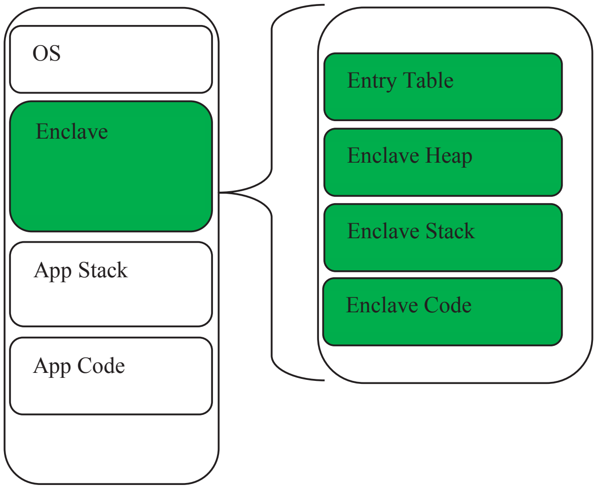

Figure 1: Enclave within Application's Virtual Address Space

2 Protection Overview

SGX prevents all other software from accessing the code and data located inside an enclave including system software and access from other enclaves. Attempts to modify an enclave's contents are detected and either prevented or execution is aborted. A summary of security properties are:

- SGX provides detection of an integrity violation of an enclave instance from software attacks and prevents access to tampered code/data upon detection.

- SGX provides confidentiality of code/data of an enclave instance from software attacks.
- SGX provides isolation between all enclave instances.
- SGX prevents replay of an enclave instance from software attacks.

In addition the hardware ensures execution starts only at enclave authorized locations and that unplanned exits from the enclave do not leak enclave information.

Finally data inside an enclave must be protected from tampering from all software outside the enclave's trust boundary, even when the enclave is sent to disk or unprotected memory by the OS or VMM managing the system resources.

To achieve these protections, several new capabilities are needed by hardware. First, code executing inside an enclave must be able to access code and data internal to the enclave while access from outside the enclave is prohibited. Secondly the translation from the application's virtual address to the correct physical address must be kept the same as when the application developer built the application[1]. More details and description of the properties needed to achieve a secure container are described in [4]. While enclave data is resident within registers, caches, or other logic blocks within the processor package, unauthorized access via software is prevented using access control mechanisms built into the processor. However, when enclave data leaves the -package caches to be written to platform memory, the data is automatically encrypted and integrity protected preventing memory probes or other techniques to view, modify, or replay data or code contained within an enclave.

3 Programming Model

SGX architecture includes 17 new instructions, new processor structures and a new mode of execution. These include loading an enclave into protected memory, access to resources via page table mappings, and scheduling the execution of enclave enabled application. Thus, system software still maintains control as to what resources an enclave can access.

An application can be encapsulated by a single enclave or can be decomposed into smaller components, such that only security critical components are placed into an enclave.

3.1 SGX Instruction Set

SGX operations can be categorized into the following functions: enclave build/teardown, enclave entry/exit, enclave security operations, paging instructions, and debug instructions. These instructions are summarized in sections below.

Enclave Build and Teardown

Table 3-1 includes the instructions which are used to allocate protected memory for the enclave, load values into the protected memory, measure the values loaded into the enclave's protected memory, and teardown the enclave after the application has completed.

Table 3-1 Enclave Build Instructions

Instruction	Description
ECREATE	Declare base and range, start build
EADD	Add 4k page
EEXTEND	Measure 256 bytes
EINIT	Declare enclave built
EREMOVE	Remove Page

Enclave Entry and Exit

Table 3-2 includes the instructions which are used to enter and exit the enclave. An enclave can be entered, EENTER, and exited, EEXIT, explicitly. It may also be exited asynchronously AEX, due to interrupts or exceptions. In the case of AEX the hardware will save all secrets inside the enclave, scrub secrets from registers, and return to external program flow.It then resumes where it left off execution.

Table 3-2 Enclave Entry and Exit Operations

Instruction	Description
EENTER	Enter enclave
ERESUME	Resume enclave
EEXIT	Leave enclave
AEX	Asynchronous enclave exit

Enclave Paging

The SGX paging instructions in Table 3-3 allow system software to securely move enclave pages to and from unprotected memory.

Table 3-3 Enclave Paging Instructions

Instruction	Description
EPA	Create version array page
ELDB/U	Load an evicted page into protected memory
EWB	Evict a protected page
EBLOCK	Prepare for eviction
ETRACK	Prepare for eviction

Enclave Debug

The enclave debug instructions in Table 3-4 allow developers to use familiar debugging techniques inside special debug enclaves. A debug enclave can be single stepped and examined. A debug enclave cannot share data with a production enclave. This protects enclave developers if a debug enclave should escape the development environment. No further details are presented on debug in this paper.

Table 3-4 Enclave Debug Instructions

Instruction	Description
EDBGRD	Read inside debug enclave
EDBGWR	Write inside debug enclave

Enclave Security Operations

The enclave security instructions in Table 3-5 allow an enclave to prove to an external party that the enclave was built on hardware which supports the SGX instruction set. Details of this process and use of key generation instruction are detailed in [1]

Table 3-5 Enclave Security Instructions

Instruction	Description
EREPORT	Enclave report
EGETKEY	Generate unique key

3.2 Address Range

The enclave executes within an application's virtual address space. An enclave is a subset of that address space. All protections are done based on the enclave's virtual address. This address is

[1] In the IA, software executes using virtual addresses but the underlying hardware uses physical addresses. The translation between the two is typically managed by privileged software which is an untrusted agent in the SGX attack model

declared by software in the ECREATE instruction

3.3 Data Structures and Components

SGX defines new data structures to maintain the information needed enforcing SGX security properties. The major structures are shown in Table 3-6.

Table 3-6 Enclave Data Structures and Components

Structure	Description
Enclave Page Cache (EPC)	Contains protected code and data in 4K pages
Enclave Page Cache Map (EPCM)	Contains meta-data of enclave page
SGX Enclave Control Store (SECS)	Meta data for each enclave
Thread Control Structure (TCS)	Meta data for each thread
VA Page	Version Array of evicted pages
SIGSTRUCT	enclave's signature structure, the sealing identity

The EPC provides the protected memory region for enclaves in the machine. The EPCM is the security meta-data attached to each EPC page. The EPCM contains the information needed by the hardware to protect the enclave memory accesses. There is a 1:1 mapping between an EPC page and an EPCM entry.

The SECS consumes 1 page of the EPC and contains meta-data needed for that particular enclave.

The TCS consumes 1 page of the EPC and contains meta-data used by hardware to control per logical processor entry into the enclave.

In additions to enclave pages, there are SGX structures that do not belong to any enclave. These structures are used for internal booking of the version counters of the evicted enclave pages. See section 7.2 for details about enclave page eviction mechanism.

Finally a SIGSTRUCT structure is passed into EINIT to provide a more flexible sealing identity and attestation [2].

3.4 Enclave Mode

When a processor enters into an enclave it begins to run in enclave mode. This mode changes the memory access semantics to perform additional checks on memory accesses. It allows the code inside an enclave to access that particular enclave. Otherwise memory accesses to the EPC result in return of abort page value.

3.5 Resource Management

The EPC is a shared system wide resource meant to be managed by privileged SW. The EPC contains the pages which are needed for execution of an enclave. An enclave does not require all of its pages to be present in the EPC to execute. This is similar to virtual memory in Intel Architecture where pages not currently executing can be moved out of memory.

4 Hardware components

To implement SGX memory protections, new hardware and structures are required. The Enclave Page Cache (EPC) is protected memory where enclave pages and SGX structures are stored. This memory is protected from hardware and software access.

Inside the EPC, code and data from many different enclaves reside. When an enclave performs a memory access to the EPC, the processor decides whether or not to allow the access. The processor maintains security and access control information for every page in the EPC in a hardware structure called the Enclave Page Cache Map (EPCM). This structure is consulted by the processor's Page Miss Handler (PMH) hardware module. The PMH mediates access to memory by consulting page tables maintained by system software, range registers, and the EPCM

4.1 Enclave Page Cache

The Enclave Page Cache (EPC) is protected memory used to store enclave pages and SGX structures. The EPC is divided into 4KB chunks called an EPC page. EPC pages can either be valid or invalid. A valid EPC page contains either an enclave page or an SGX structure. The security attributes for each EPC page are held in an internal micro-architecture structure called EPCM, which is described below.

Each enclave instance has an enclave control structure, SECS. Every valid enclave page in the EPC belongs to exactly one enclave instance. System software is required to map enclave virtual addresses to a valid EPC page.

4.2 Enclave Page Cache Map

The Enclave Page Cache Map (EPCM) is a protected structure used by the processor to track the contents of the EPC. The EPCM is comprised of a series of entries with exactly one entry for each page in the EPC. The EPCM is managed by the processor as part of various SGX instructions and is never directly accessible to software or to devices. The format of the EPCM is micro-architectural and is implementation dependent. However, logically, each EPCM entry holds the following information:

- Whether the EPC page is valid or invalid
- The enclave instance that owns the page.
- The type of page (REG, TCS, VA, SECS)
- The virtual address through which the enclave is allowed to access the page
- The enclave specified read/write/execute permissions on that page
- Whether the page is accessible or not (BLOCKED or UNBLOCKED). See section 7.1.

The EPCM structure is used by the CPU in the address-translation flow to enforce access-control on the enclave pages loaded into the EPC. Logically it provides an additional secure layer of access control in addition to "legacy" segmentation, paging tables and extended paging tables mechanisms.

4.3 EPC Layout

EPC layout is specific to a particular implementation, and is enumerated through the CPUID instruction [5].

At a high level, a CPU that supports SGX and implements EPC in cryptographically protected platform DRAM supports the ability for the BIOS to reserve a range(s) of memory called Processor Reserved Memory (PRM). The BIOS allocates the PRM by configuring a set of range registers, collectively known as the PRMRR.

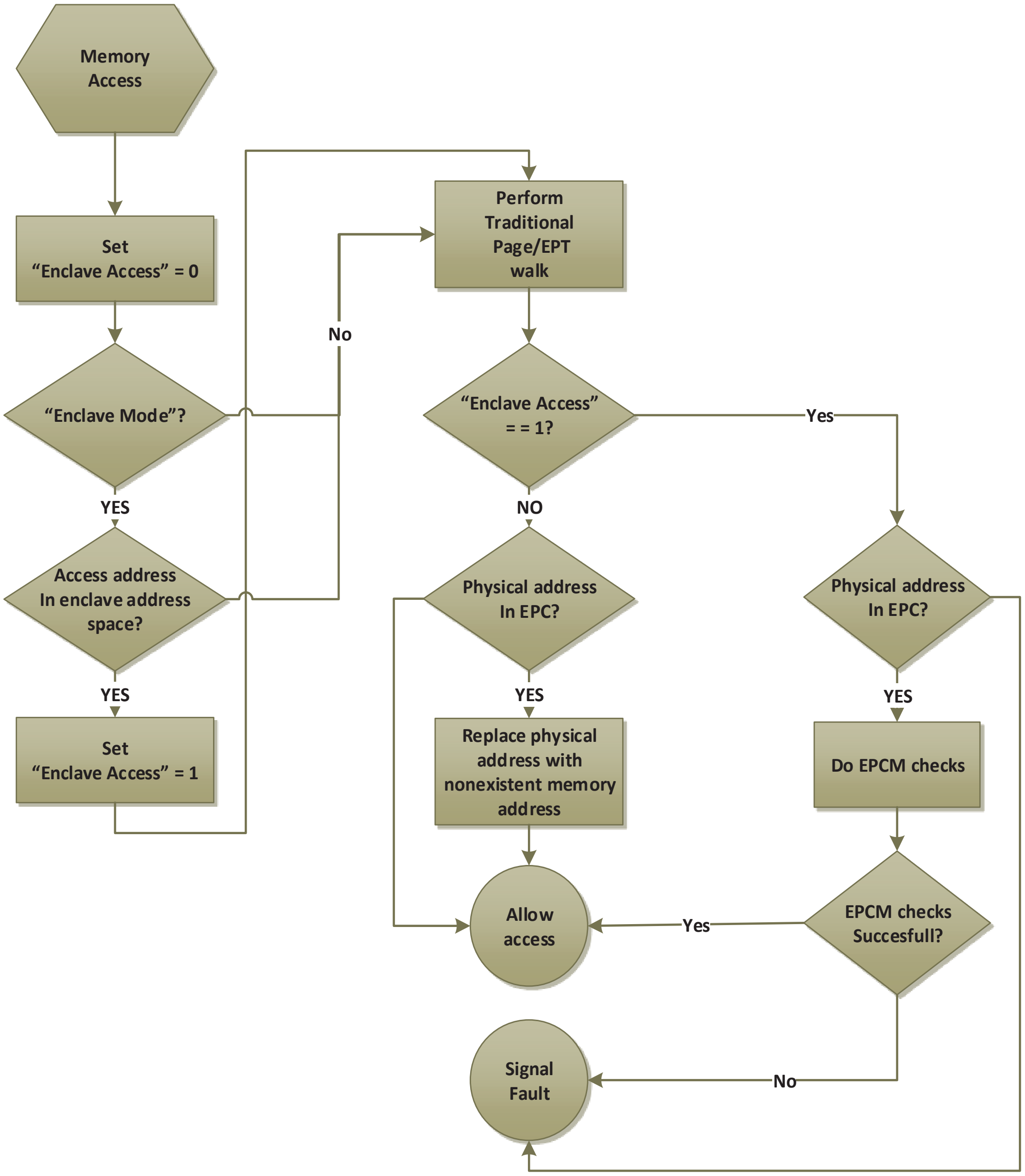

Figure 2 SGX Enclave Access Check

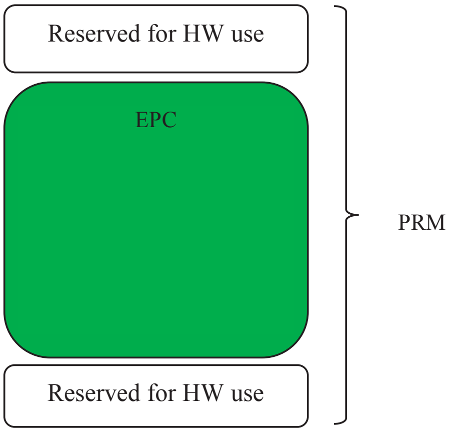

Figure 3 PRM Layout

An example layout is shown in Figure 3The exact layout of the PRM and EPC is model-specific, and depends on BIOS settings.

4.4 EPC Memory Protection

This section describes mechanisms employed by the CPU in order to protect EPC memory.

Memory Encryption Engine

Use of main memory as storage for the EPC is very desirable for many implementations. The challenge is there are many known software and hardware attacks that can be waged on DRAM memory. Cryptographically protecting the EPC contents in DRAM is one way to defend against these attacks.

To that end, the Memory Encryption Engine (MEE) is a hardware unit that encrypts and integrity protects selected traffic between the processor package and the main memory (DRAM). The overall memory region that an MEE operates on is called an MEE Region. Depending on implementation, the PRM is covered by one or more MEE regions.

Memory Access Semantics

CPU memory protection mechanisms physically block access to PRM from all external agents (DMA, graphic engine, etc.), by treating such accesses as references to non-existent memory. To access a page inside an enclave using MOV and other memory related instructions, the hardware checks as described in Figure 2, the following:

- Logical processor is executing in “enclave mode”
- Page belongs to enclave that the logical processor is executing
- Page accessed using the correct virtual address

If the accessed page is not part of the enclave’s virtual address space but is part of the EPC then the access is treated as a reference to nonexistent memory. If the page is outside of the enclave virtual address space, then hardware allows the enclave code to access the memory outside of PRM. If the page is outside of the enclave’s virtual address space and resolves into a PRM page, hardware prevents such access by signaling a fault. Accesses by a processor not in enclave mode to an enclave page are treated as a reference to nonexistent memory.

5 Enclave creation process

The enclave creation process loads enclave binary into the EPC and establishes the enclave identity.

The enclave creation process is divided into multiple stages: initialization of enclave control structure, allocation of EPC pages and loading of enclave content into the pages, measurement of the enclave contents and finally establishing the enclave identity. These steps are supported by the following instructions: ECREATE EADD, EEXTEND, and EINIT.

ECREATE starts the enclave creation process and initializes the SGX Enclave Control Structure (SECS) which contains global information about the enclave. EADD commits EPC pages to an enclave and records the commitment but not the contents in the SECS. The memory contents of an enclave are explicitly measured by EEXTEND. EINIT completes the creation process which finalizes the enclave measurement and establishes the enclave identity. Until an EINIT is executed, enclave entry is not permitted.

5.1 Starting Enclave Creation

The enclave creation process begins with ECREATE which converts a free EPC page into an SECS page and initializes the structure. As part of ECREATE, system software selects which EPC page to be made an SECS and specifies several attributes of the enclave including the range of protected addresses the enclave can access, the mode of operation (32bit vs 64 bit), processor features supported by the enclave, and finally whether debug access is allowed.

5.2 Adding Pages and Measuring the Enclave

Once the SECS has been created, enclave pages can be added to the enclave via EADD. This involves converting a free EPC page into either a REG or a TCS.

EADD when invoked will initialize the EPCM entry to indicate the type of page (REG or TCS), the linear address that the enclave will access the page, the enclave RWX permissions for the page, and associates the page to the SECS provided as input. The EPCM entry information is used by hardware to provide SGX access control to the page as discussed in section 4. EADD will then record EPCM information in a cryptographic log stored in the SECS and copy 4 K bytes of data from unprotected memory to the allocated EPC page.

System software is responsible for selecting a free EPC page, the type of page to be added, the attributes the page, the contents of the page, and the SECS (enclave) to which the page is to be added.

After a page has been added to an enclave, software can measure a 256 byte region as determined by the developer by invoking EEXTEND. Thus to measure an entire page, system software must execute EEXTEND 16 times. Each invocation of EEXTEND adds to the cryptographic log, a header indicating which region is being measured followed by the 256 bytes of information.

Entries in the cryptographic log define the measurement of the enclave and are critical in gaining assurance that the enclave was correctly constructed by the untrusted system software. Examples of incorrect construction includes adding multiple pages with the same enclave linear address resulting in an alias, loading modified contents into an enclave page, or not measuring all of the enclave. Due to the potential size of the log, only a cryptographic hash of the log is actually stored in the SECS. Correct construction results in the cryptographic log matching the one built by the enclave owner in SIGSTRUCT. It can be verified by the remote party. [2]

5.3 Initializing an Enclave

Once system software has completed the process of adding and measuring pages, the enclave needs to be initialized. Initializing an enclave prevents the addition and measurement of enclave pages and enables enclave entry. The initialization process finalizes the cryptographic log and establishes the enclave identity and sealing identity used by EGETKEY and EREPORT.

The sealing identity is managed by a sealing authority represented by the hash of a public key used to sign a structure processed by EINIT. The sealing authority assigns a product ID and security version number to a particular enclave identity comprising the attributes of the enclave and the measurement of the enclave.

EINIT establishes the sealing identity using the following steps:

1. Verifies that SIGSTRUCT is signed using the public key enclosed in the SIGSTRUCT

2. Checks that measurement of the enclave matches the measurement of the enclave specified in SIGSTRUCT

3. Checks that the enclave's attributes are compatible with those specified in SIGSTRUCT

4. Finalizes the measurement of the enclave and records the sealing identity and enclave identity (the sealing authority, product id and security version number) in the SECS

If EINIT was successful it enables the enclave to be entered.

6 Enclave entry and exiting

Critical to preserving the integrity of an enclave is to control transfer of execution into and out of an enclave. The entry process needs to clear any cached translations that overlap with the enclave's protected address region. This ensures that all protected enclave memory accesses are properly checked. The entry process must identify where inside the enclave the processor should transfer control and enable enclave mode of execution. Exiting an enclave must again clear any cached translations referring to the enclave's protected address region so that no other software can use the cached translations to access the enclave's protected memory.

While operating in enclave mode, an interrupt, fault or exception may occur. Traditionally, the processor would vector to a fault handler specified by system software. The fault handler saves the register state and services the event. Once the event has been serviced, system software restores the register state and returns control to where software was interrupted. Allowing system software to read and/or modify the register state of an enclave places system software within the trust boundary of the enclave. Consequently, SGX introduces a new routine to protect the integrity and confidentiality of the enclave.

SGX offers the EENTER and EEXIT instructions to enter and exit an enclave programmatically (e.g. as part of call/return sequence). When enclave exit occurs due to an event, the processor invokes a special internal routine called Asynchronous Exit (AEX) which saves the enclave register state, clears the registers, sets the faulting instruction address to a value specified by EENTER. The ERESUME instruction restores the state back to allow the enclave to resume execution.

6.1 Synchronous Entry and Exit

The EENTER instruction is the method to enter the enclave under program control. To execute EENTER, software must supply an address of a TCS that is part of the enclave to be entered. The TCS indicates the location inside the enclave to transfer control and where inside the enclave AEX should store the register state. When a logical processor enters an enclave, the TCS is considered busy until the logical processors exits the enclave. SGX allows an enclave builder to define multiple TCS structures, thereby providing support for multithreaded enclaves

EENTER also defines the Asynchronous Exit Pointer(AEP) parameter. AEP is an address external to the enclave which is used to transition back into the enclave after an AEX. The AEP is the address an exception handler will return to using IRET. Typically the location would contain the ERESUME instruction. ERESUME transfers control to the enclave address retrieved from the enclave saved state.

EENTER performs the following operations:

1. Check that TCS is not busy and flush TLB entries for enclave addresses

2. Change the mode of operation to be in enclave mode

3. Save the stack pointer, RSP, and frame pointer,RBP, for later restore on enclave asynchronous exit

4. Save the thread's OS supported state (XCR0) and replace it with the subset supported by the enclave (XFRM)

5. Save the AEP away for possible AEX

6. Transfer control from outside enclave to location inside the enclavedefined by the TCS

The EEXIT instruction is the method of leaving the enclave under program control, it performs the following operations:

1. Clear enclave mode and TLB entries for enclave addresses

2. Mark TCS as not busy

3. Transfer control from inside the enclave to a location on the outside specified by register, RBX

6.2 Asynchronous Exit (AEX)

Asynchronous events, such as exceptions and interrupts may occur during execution inside an enclave. These events are referred to as Enclave Exiting Events (EEE). Upon an EEE, the processor state is securely saved inside the enclave and then replaced by a synthetic state to prevent leakage of secrets. The process of securely saving state and establishing the synthetic state is called an Asynchronous Enclave Exit (AEX).

As part of the EEE the AEP is pushed onto the stack as the location of the faulting address. This is the location where control will return after executing the IRET. The ERESUME can be executed from that point to reenter the enclave.

After AEX has completed, the logical processor is no longer in enclave mode and the exiting event is processed normally. Any new events that occur after the AEX has completed are treated as having occurred outside the enclave (e.g. a #PF in dispatching to an interrupt handler).

6.3 Resuming Execution after AEX

After system software has serviced the event that caused the logical process to exit an enclave, the logical processor can re-start execution using ERESUME. Unlike EENTER, ERESUME restores registers and returns control to where execution was interrupted. If the cause of the exit was an exception or a fault and was not resolved, then the event will be triggered again. For example, if an enclave performs a divide by 0 operation, executing ERESUME will cause the enclave to attempt to re-execute the faulting instruction. In order to handle an exception that occurred inside the enclave, software should enter the enclave at a different location and invoke an exception handler, the EENTER instruction should be used. The exception handler can attempt to resolve the faulting condition or simply return and indicate to software that the enclave should be terminated.

7 EPC paging

Allowing system software to oversubscribe the EPC increases the number of protected applications that can be supported concurrently. The SGX architecture offers instructions to allow system software to oversubscribe the EPC by securely evicting and loading enclave pages and SGX structures.

The contents of an enclave page evicted from the EPC to main memory must have the same level of integrity, confidentiality and replay protection as when the contents resided within the EPC.

To achieve this objective, the paging instructions enforce the following rules:

1. An enclave page must be evicted only after all cached translations to that page have been evicted from all logical processors
2. The contents of the evicted enclave page must be encrypted before being written out to main memory
3. When evicted enclave page is reloaded into EPC it must have identical page type, permissions, virtual address, content, and be associated to the same enclave as at the time of eviction
4. Only the last evicted version of an enclave page can be allowed to be reloaded

7.1 Preparing an enclave page for eviction

To prepare the enclave page for eviction, system software marks the page to be evicted as BLOCKED using the EBLOCK instruction. Once an EPC page has been marked as BLOCKED, the processor prevents any new Translation Lookaside Buffer, TLB, entries that map that EPC page from being created. However, TLB entries that reference this page may exist in one or more logical processors. These TLB entries must be removed before the page can be removed from the EPC. In SGX this must be guaranteed by hardware. While only the TLB entries for the page must be removed, we chose a simpler implementation option. In this implementation all TLB entries for that particular enclave are removed.

TLB entries created during enclave execution are evicted when exiting the enclave. Thus an enclave page that is BLOCKED can be safely evicted after all logical processors that were executing inside the enclave to which the page belongs have exited the enclave at least once since the EBLOCK.

The ETRACK instruction is used to configure micro architectural trackers to detect when all logical processors executing in an enclave at the time of executing the ETRACK instruction have exited the enclave.

7.2 Evicting the enclave page

System software uses EWB to evict an enclave page that has been prepared for eviction (blocked and no TLB entries referring to the page). The system software must also allocate a VA page entry to hold the version counter to be associated with this page. The EWB evicts a page from EPC by:

1. Assigning a unique version value for the page and recording it in the VA page entry allocated by the system software

2. Encrypting the EPC page using the paging encryption key

3. Computing a cryptographic MAC over the encrypted page contents, version counter and the additional metadata for the EPC page

4. Writing out the encrypted page contents and the metadata along with the computed MAC to the main memory buffers passed to the EWB instruction as parameters

The system software must retain the encrypted page contents, the metadata and the VA entry with this EPC page in order to reload it back into EPC.

7.3 Reloading an evicted page

System software uses ELDU or ELDB to reload an evicted enclave page into the EPC. The system software allocates a free page in the EPC and passes the encrypted page contents, the metadata generated at eviction and the VA entry used to evict the page as parameters to the ELDU/ELDB instructions. The ELDU & ELDB instructions are identical except that on successful execution of the ELDB instruction the EPC page used to reload the enclave page is marked as BLOCKED in the EPCM.

The ELDU/ELDB instructions reload the enclave page using below steps:

1. Copy the encrypted enclave page contents to the allocated EPC page

2. Verify the MAC on the metadata, version counter from the specified VA entry and encrypted enclave page contents

3. If verification succeeds, decrypt the enclave page contents into the EPC page allocated by system software and clear the VA entry to prevent any future replay attempts

4. Update the EPCM associated with the EPC page with the attributes from the metadata

7.4 Evicting the VA Page

In order to allow scaling of the EPC and software to build extremely large enclaves, the pages containing versions must also be pageable. VA pages can be evicted. In the limit, one anchor page is required to allow VA pages to be loaded back into the enclave.

The system software may evict VA pages from the EPC using EWB. The VA page to be evicted is assigned a version number and the version number is recorded in a VA entry in a second VA page. Prior to reloading any evicted enclave pages the system software is required to reload the VA page containing the VA entry.

8 Summary and related work

Protecting software and secrets on commercially available processors has been elusive goal for many years. Approaches to solve the problem by adding a new layer of system software have had limited deployment. Alternatively, closed systems often have restricted user choice with regard to the system software and applications that may be loaded onto a platform.

A number of research projects have developed secure processor architectures to protect software. These include XOM [6], AEGIS [7], SP [8], Bastion [9], and HyperWall [10]. XOM, AEGIS, and Bastion are architecture additions to protect trusted software modules from the operating system. AEGIS like SGX requires no trusted system software but differs in the memory protection scheme. XOM and Bastion both include a trusted hypervisor with the trusted computing base of the trusted software modules. Bastion, for example, depends on a trusted hypervisor to check that the OS has properly mapped trusted modules to protected memory. HyperWall provides protection at the virtual machine granularity which mean that trusted applications have an operating system within their trust boundary.

Finally [11] is recent work which provides very similar concepts to early work on SGX [12]. SGX now uses a different integrity mechanism which provides more flexibility for OS and VMMs.

In an era where software and services are deployed over the internet, SGX enables service providers to provision applications remotely and to know with confidence that their secrets are protected. This paper describes the SGX instructions and hardware extensions to load an enclave into protected memory, establish the enclave's identity, enter and exit an enclave, and oversubscribe SGX protected memory while requiring trust in only in the enclave application.

9 ACKNOWLEDGEMENTS

The authors of this paper wish to acknowledge the contributions of many hardware and software architects and designers who have worked in developing this innovative technology.

Intel is a trademark of Intel Corporation in the U.S. and/or other countries.

10 REFERENCES

[1] Ponemon Institute, "2012 Cost of Cyber Crime Study:," Ponemon Institute, 2012.

Cited [2] I. Anati, S. Gueron, S. Johnson and V. Scarlata, "Innovative Technology for CPU Based Attestation and Sealing," in *ISCA-HASP*, Tel Aviv, 2013.

Cited [3] M. Hoekstra, R. Lal, P. Pappachan, C. Rozas, V. Phegade and J. Del Cuvillo, "Using Innovative Instructions to Create Trustworthy Software Solutions," in *ISCA-HASP*, Tel-Aviv, 2013.

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

[5] Intel Corp, "http://www.intel.com/content/www/us/en/processors/architectures-software-developer-manuals.html," Intel, June 2013. [Online]. Available: http://download.intel.com/products/processor/manual/325462.pdf.

[6] 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.

[7] 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.

Cited [8] 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.

Cited [9] D. Champagne and R. Lee, "Scalable architectural support for trusted software," in *16th International Symposium on High Performance Computer Architecture (HPCA)*, 2010.

Pass [10] 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.

Cited [11] Rick Boivie, IBM Corp, "Secure Blue++: CPU Support for Secure Execution," IBM Watson Research Center, Yorktown Heights NY, 2012.

Cited [12] F. MCKEEN, U. SAVAGAONKAR, C. ROZAS, M. GOLDSMITH, H. HERBERT, A. ALTMAN, 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.

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

A "Mission Critical Application" is any application in which failure of the Intel Product could result, directly or indirectly, in personal injury or death. SHOULD YOU PURCHASE OR USE INTEL'S PRODUCTS FOR ANY SUCH MISSION CRITICAL APPLICATION, YOU SHALL INDEMNIFY AND HOLD INTEL AND ITS SUBSIDIARIES, SUBCONTRACTORS AND AFFILIATES, AND THE DIRECTORS, OFFICERS, AND EMPLOYEES OF EACH, HARMLESS AGAINST ALL CLAIMS COSTS, DAMAGES, AND EXPENSES AND REASONABLE ATTORNEYS' FEES ARISING OUT OF, DIRECTLY OR INDIRECTLY, ANY CLAIM OF PRODUCT LIABILITY, PERSONAL INJURY, OR DEATH ARISING IN ANY WAY OUT OF SUCH MISSION CRITICAL APPLICATION, WHETHER OR NOT INTEL OR ITS SUBCONTRACTOR WAS NEGLIGENT IN THE DESIGN, MANUFACTURE, OR WARNING OF THE INTEL PRODUCT OR ANY OF ITS PARTS.

Intel may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "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.

Intel®, the Intel® Logo, Intel® Inside, Intel® Core™ , Intel® Atom™, and Intel® Xeon® are trademarks of Intel Corporation in the U.S. and/or other countries. Other names and brands may be claimed as the property of others.