Introduction to Intel ® Software Guard Extensions (SGX) and SGX Virtualization

Kai Huang
OTC, Intel Corporation

Legal Disclaimer

No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document.

Intel disclaims all express and implied warranties, including without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement, as well as any warranty arising from course of performance, course of dealing, or usage in trade.

This document contains information on products, services and/or processes in development. All information provided here is subject to change without notice. Contact your Intel representative to obtain the latest forecast, schedule, specifications and roadmaps.

The products and services described may contain defects or errors known as errata which may cause deviations from published specifications. Current characterized errata are available on request.

Copies of documents which have an order number and are referenced in this document may be obtained by calling 1-800-548-4725 or by visiting www.intel.com/design/literature.htm.

Intel and the Intel logo are trademarks of Intel Corporation or its subsidiaries in the U.S. and/or other countries.

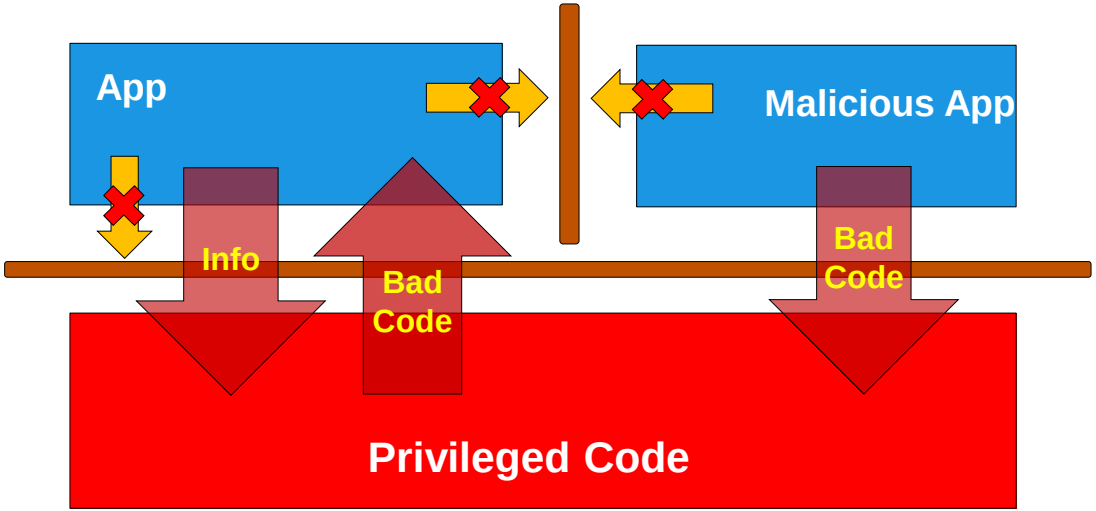
*Other names and brands may be claimed as the property of others

Agenda

- **SGX Introduction**
- **SGX Virtualization**
- **Backup**

The Basic Issue: Too Big TCB

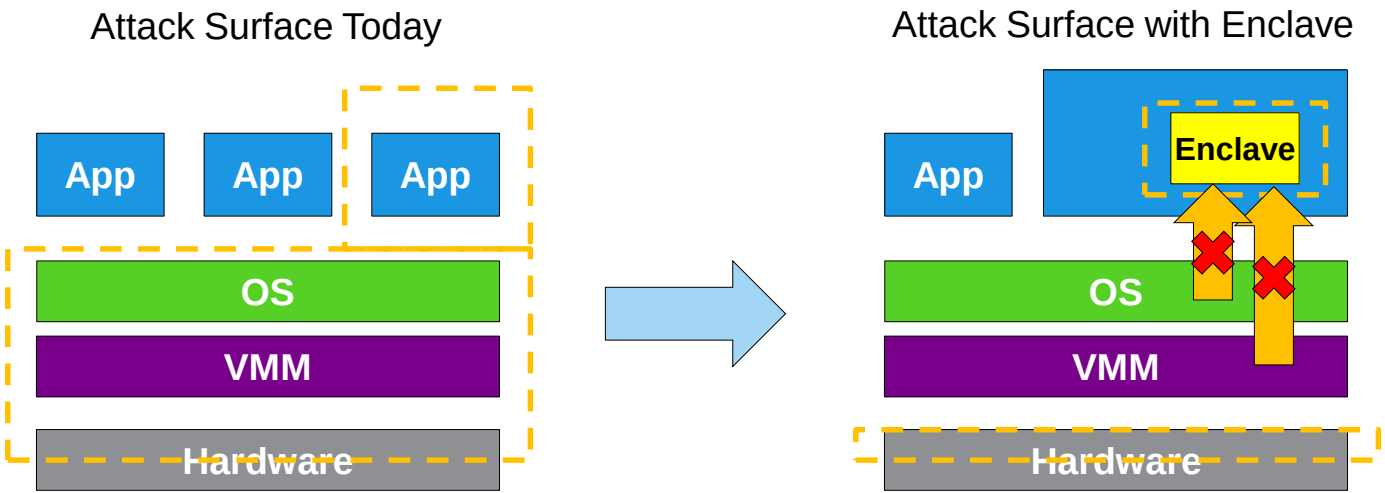
HW protects apps from OS, and apps from each other, until



privileged code got compromised…

Apps not protected from privileged code attacks

SGX: Reduce TCB to “HW + Enclave”



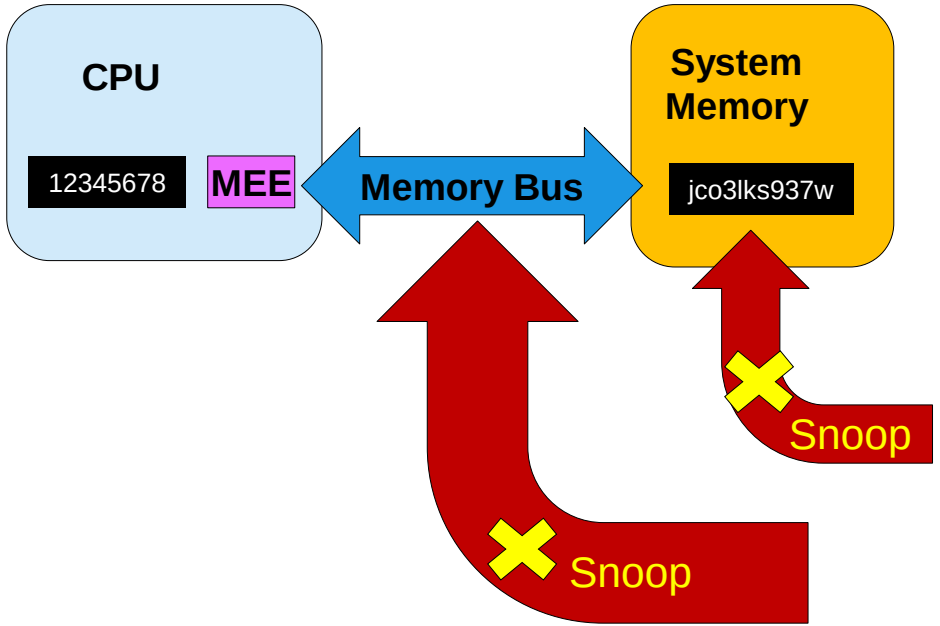
Enclave:

- A protected container in App’s address space (ring 3).
- Even privileged SW cannot access enclave directly.
- Reduce TCB to “HW + Enclave”

➔ ***App gets its own capability of protection***

SGX: Prevent Memory Snooping Attacks

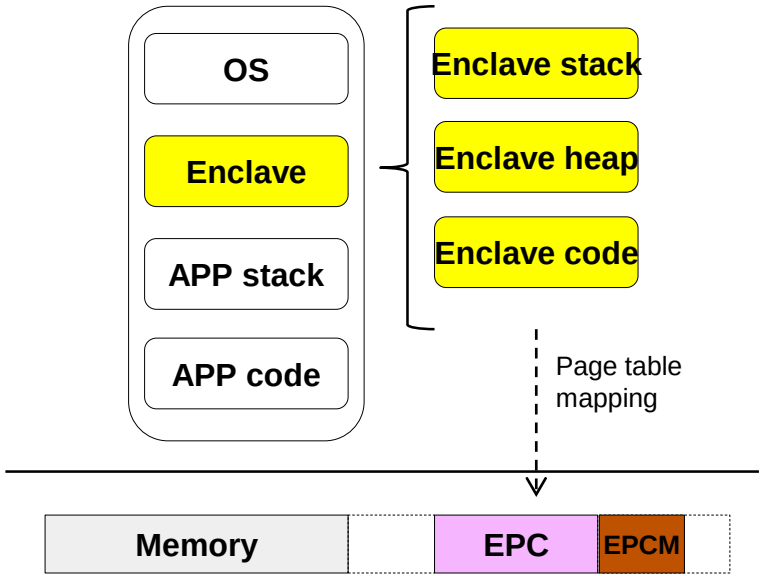
- Security perimeter is the CPU package boundary
- Data and code unencrypted inside CPU package
- Data outside CPU package is encrypted and/or integrity checked
- External memory reads and bus snooping only see encrypted data



MEE: SGX Memory Encryption Engine

SGX Enclave

- **Enclave**
  - Trusted Execution Environment embedded in application
  - Provides confidentiality and/or integrity
  - With it's own code/data.
  - With controlled entry points
  - Multiple threads supported
- **EPC (Enclave Page Cache)**
  - Trusted Memory to commit enclave (via page table)
  - With additional access check
  - Typically reserved by BIOS as Processor Reserved Memory
    - Along with EPCM with limited size (ex, 32M, 64M, 128M)
- **New SGX instructions to manage/access Enclave**
  - ENCLS, ENCLU; ENCLV

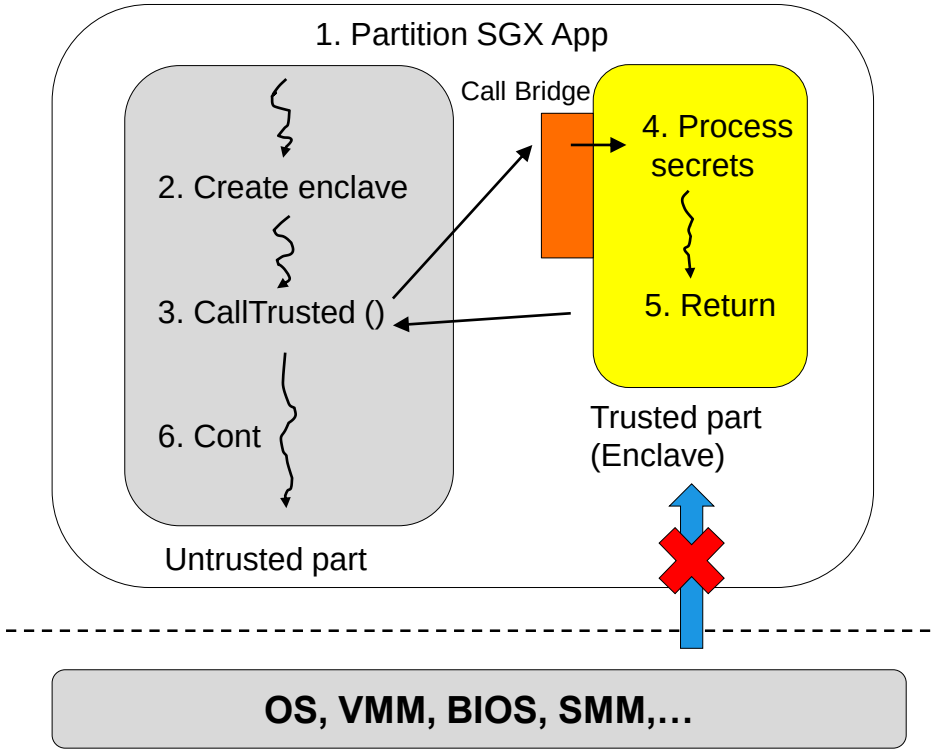


*** EPCM (Enclave Page Cache Map) :**

Used by HW to track EPC status (not-visible to SW)

SGX Application Flow

1. Define and partition App to untrusted and trusted part.
2. App creates enclave
3. Trusted function is called;
4. Code in enclave process secrets.
5. Trusted function returns.
6. App continues normal execution.



Instruction Behavior Changes in Enclave

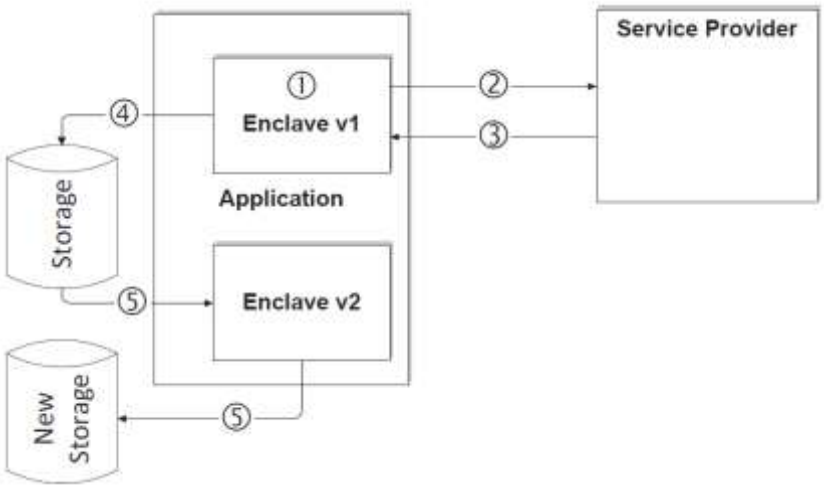
- Invalid Instructions

Instructions	Result	Comment
CPUID, GETSEC, RDPMC, SGDT, SIDT, SLDT, STR, VMCALL, VMFUNC	#UD	Might cause VM exit.
IN, INS/INSB/INSW/INSD, OUT, OUTS/OUTSB/OUTSW/OUTSD	#UD	I/O fault may not safely recover. May require emulation.
Far call, Far jump, Far Ret, INT n/INTO, IRET, LDS/LES/LFS/LGS/LSS, MOV to DS/ES/SS/FS/GS, POP DS/ES/SS/FS/GS, SYSCALL, SYSENTER	#UD	Access segment register could change privilege level.
SMSW	#UD	Might provide access to kernel information.
ENCLU[EENTER], ENCLU[ERESUME]	#GP	Cannot enter an enclave from within an enclave.

- Behavior Changes
  - RDTSC, RDTSCP: Only legal when SGX2 is available.
  - RDRAND, RDSEED, PAUSE: May cause VMEXIT
  - INVD: #UD in enclave
  - INT3

SGX Enclave Use Case

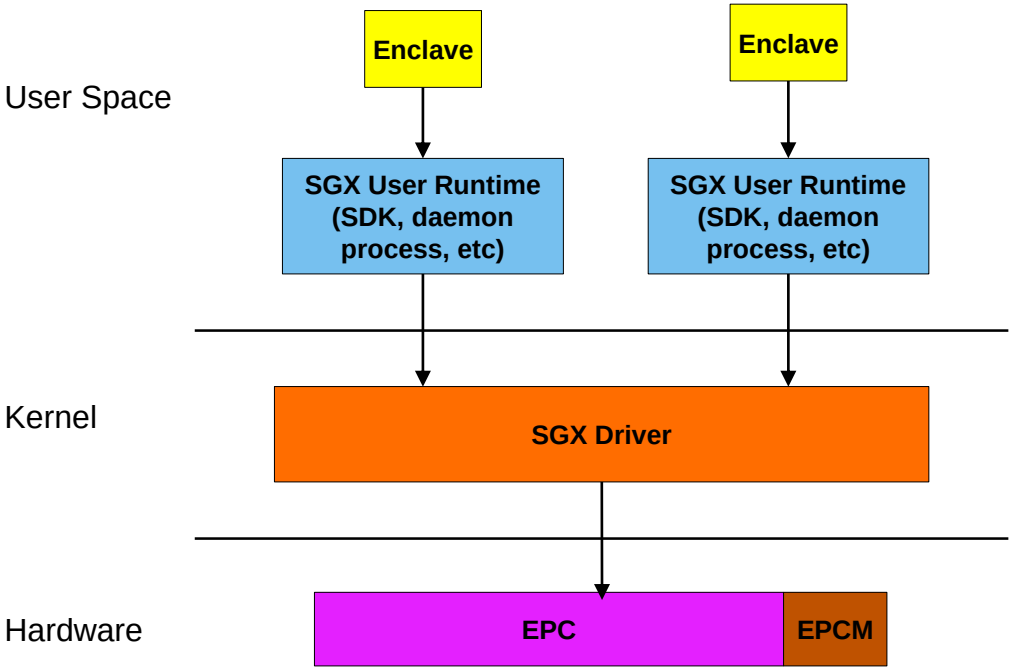
1. Enclave lunch:
   - Build and launch enclave
2. Attestation
   - Enclave proves its identity to SP.
3. Provisioning
   - SP provides secrets to enclave.
4. Sealing/Unsealing
   - Enclave derives sealing key
   - Enclave securely exports secrets for further use by using sealing key
5. Software Upgrade
   - From time to time enclave gets upgraded
   - Unseal exported secrets using old sealing key
   - Exports secrets using new sealing key



SGX Features – Functionality's Perspective

- **SGX 1**
  - Basic Enclave support
  - EPC Eviction & Reload
- **SGX 2 (Enclave Dynamic Memory Management, EDMM)**
  - malloc for EPC; Runtime permission change.
- **Launch Control Policy**
  - Support Launch Enclave (LE) signed by 3rd party.
- **VMM Oversubscription**
  - New instructions to simply VMM EPC Oversubscription

SGX High-Level HW/SW Architecture On Bare Metal



ENCLU
- EENTER, EEXIT, ERESUME
- EGETKEY
- EREPORT
- EACCEPT, EACCEPTCOPY, EMODPE

ENCLS
- ECREATE, EADD, EEXTEND, EINIT, EREMOVE
- EBLOCK, ETRACK, EWB, ELDU/ELDB
- EPA
- EAUG, EMODPR, EMODT

EPC & EPCM
- EPC: Limited resource; Managed in 4K page
- EPCM: used by HW to track EPC status (not-visible to SW)

SGX In Depth

- Enclave Lifecycle
- SGX Launch Control
- VMM Oversubscription
- SGX Attestation
- SGX Sealing

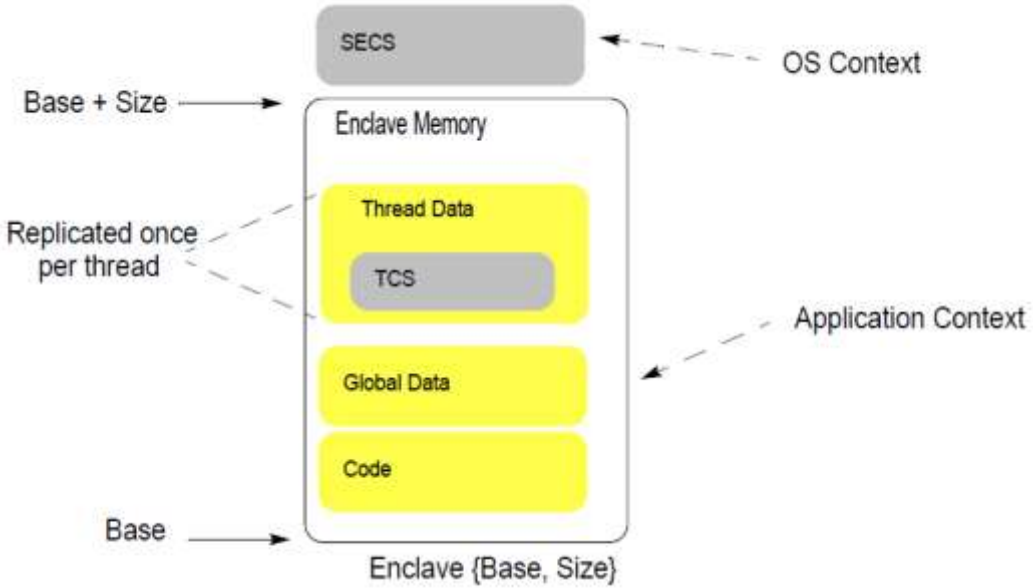
Construct & Destroy Enclave

Construct Enclave

1. **ECREATE**
   - Receive Enclave Base, size, attributes, …
   - Initialize above info to **SECS** EPC page
2. **For each Enclave page**
   - **EADD** code/data to an free EPC page
   - **EEXTEND** the content of EPC page
3. **EINIT**
   - Finalize confidentiality and integrity with **SIGSTRUCT, EINITTOKEN** and **SECS**

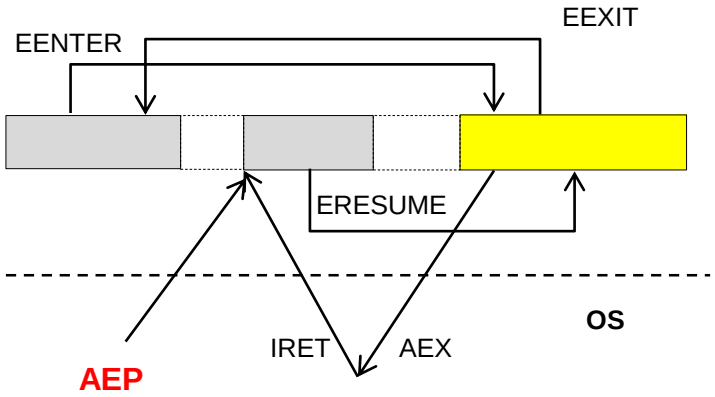
Destroy Enclave

1. **EREMOVE** all EPC pages in enclave
2. **EREMOVE** SECS



Enclave Entry & Exit

- **Synchronous Entry and Exit**
  - EENTER, EEXIT
- **Asynchronous Exit (AEX)**
  - Interrupt, Exception, VMEXIT, SMI,…
  - **ERESUME** to return to Enclave
  - Resume from AEX via **AEP**



(Asynchronous Exit Pointer)

- Special routine outside of enclave
- AEP set in EENTER as parameter
- AEP is saved to stack in AEX
- IRET returns to AEP
- ERESUME is called in AEP

No code change in OS/VMM code to support AEX!

EPC Eviction & Reload

Evict EPC page in Enclave

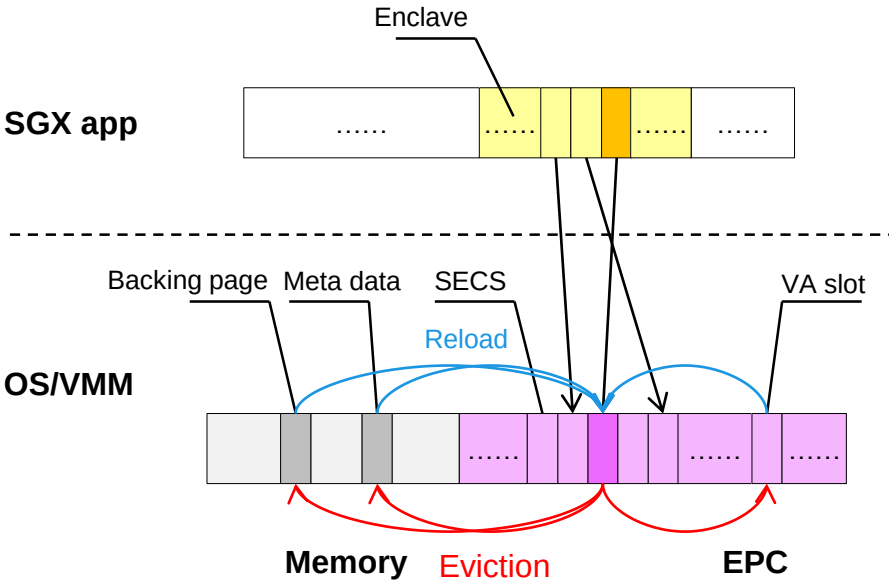
1. Select an available **VA* slot** (8 bytes in EPC)
2. Remove mapping of the EPC page
3. Send **IPIs** to kick processors out of Enclave.
4. **EBLOCK** the EPC page
5. **ETRACK** of the enclave
6. **EWB** the EPC page

Reload EPC page to Enclave

1. **ELDU** the EPC page
2. Create mapping for the EPC page

Different steps for Evicting SECS and VA page.

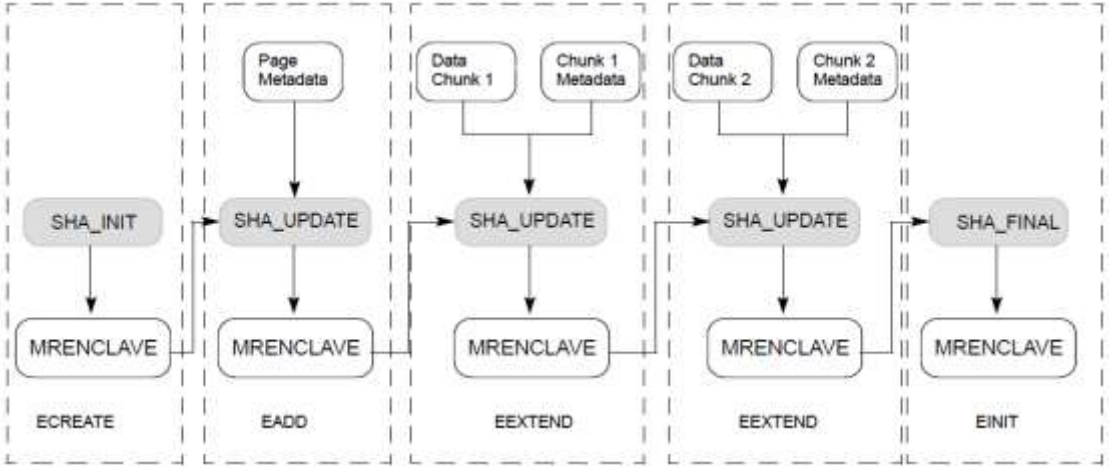
- Need to evict all Enclave page before evicting SECS.
- No EBLOCK/ETRACK required for evicting VA page.



* VA (Version Array) page is an EPC page created by **EPA**.

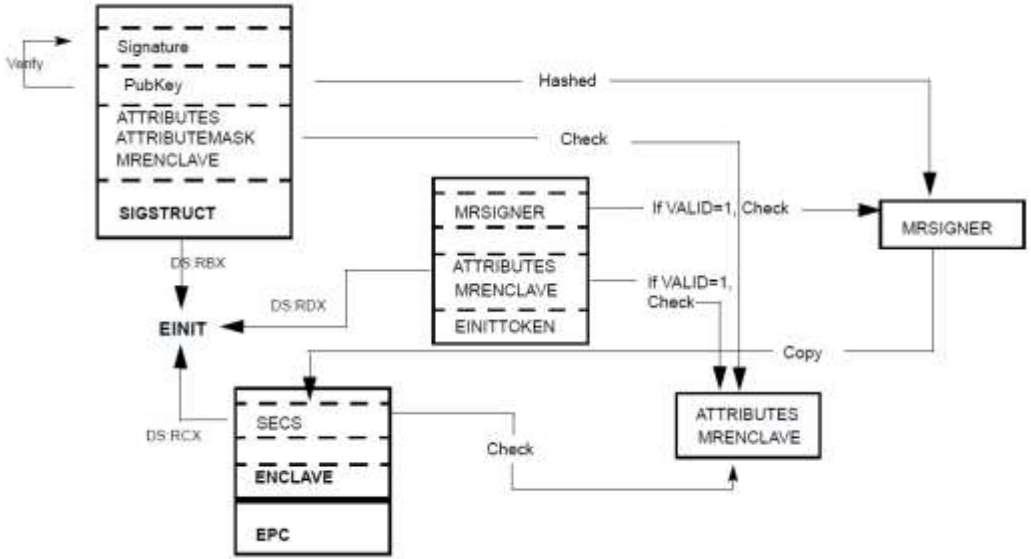
Enclave Measurement

- MRENCLAVE: cryptographic log generated during Enclave construction:
  - Content: code, data, stack, heap
  - Location of each page within enclave
  - Others: security flags being used, etc.



Establishing Enclave Identity in EINIT

- Enclave *authority* (signer) provides Enclave *certificate* (SIGSTRUCT)
  - Enclave measurement
  - Attributes
  - ISVPRODID & ISVSVN
  - RSA Pubkey
  - RSA Signature
- LE governs other enclaves by issuing EINITTOKEN
- HW verifies SIGSTRUCT and EINITTOKEN, and commit to SECS
  - MRENCLAVE
  - MRSIGNER
  - ISVPRODID & ISVSVN



MRENCLAVE: 256 bit hash of enclave measurement
MRSIGNER: 256 bit hash of signer's pubkey

SGX Launch Control

- **Launch Enclave (LE)**
  - LE is first enclave to run before running any other enclaves
  - Generates EINITTOKEN for other enclaves, thus providing control over other enclaves.
- **On SKL SGX can only run Intel's Launch Enclave.**
  - Not appreciated by open source community
- **New IA32_SGXLEPUBKEYHASH[0-3]**
  - Allow BIOS to set hash of 3rd party's RSA key
  - EINIT only runs successfully if hash of LE equals to IA32_SGXLEPUBKEYHASHn
- **New LE_WR bit (bit 17) on IA32_FEATURE_CONTROL**
  - Indicate whether IA32_SGXLEPUBKEYHASHn is writable after FEATURE_CONTROL is locked.
  - The availability of LE_WR is enumerated via CPUID.0x7.0x0:ECX[bit 2]

VMM EPC Oversubscription -- Problem

- **To support EPC oversubscription, VMM needs to know below from guests**
  - EPC page type (SECS, VA, regular, etc), status (blocked, etc)
  - SECS location
- **Without hardware support, VMM needs to trap ENCLS for above info**
  - Maintain all EPC & Enclave info from all VMs upon ENCLS VMEXIT.
  - VMM runs ENCLS on behalf of guest, and emulate result
    - VMM needs to reconstruct/remap ENCLS parameter (from guest VA to VMM VA).
- **VMM Oversubscription hardware feature (see next)**
  - Dedicated to solve above problem, thus simplifies implementation.

VMM EPC Oversubscription Hardware Feature

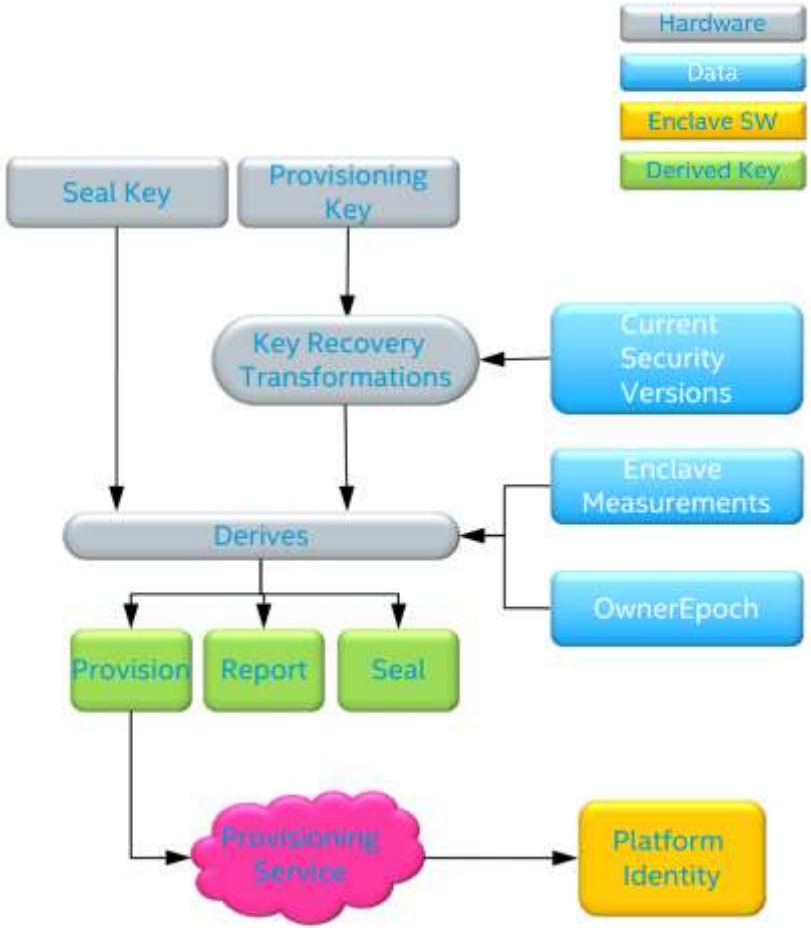
- **New ERDINFO (ENCLS) to read EPC status & info**
  - STATUS: whether SECS has any child & virtual child
  - FLAGS: Page Type (SECS, VA …) , R/W/X, BLOCKED, …
  - ENCLAVECONTEXT: GFN of SECS
    - Which is saved by ELDU/EADD running in guest during address translation.
- **New ENCLV (can only be called in VMX operation)**
  - **EINCVIRTCHILD** to increase SECS's virtual child
    - Called before EWB to prevent guest evicts SECS
  - **EDECVIRTCHILD** to decrease SECS's virtual child
    - Called after ELDU
  - **ESETCONTEXT** to save SECS's GFN after VMM reloads SECS
    - ELDU running in VMM cannot find SECS's GFN during address translation.

SGX Attestation

- Enclave proves it’s trustworthiness to verifier
  - Hardware TCB
  - Software TCB
- SGX Attestation
  - Local attestation
  - Remote attestation

SGX Key Hierarchy

- 2 platform specific root keys for key derivation
  - Provisioning key: known to Intel
  - Sealing key: unknown to Intel
- HW Current Security Version based key derivation to revoke compromised (old) keys after chip security update
- Enclave uses EGETKEY to get platform dependent keys for different purposes
  - Provision; Report; Seal; …
  - Some keys are bound to OwnerEpoch and/or Enclave identity/authority/version, etc.



SGX Local Attestation

Source enclave calls EREPORT leaf which

- Populate REPORT with identity information of calling enclave
- Derive the Report Key of Target Enclave
- Computes MAC over the REPORT using derived key

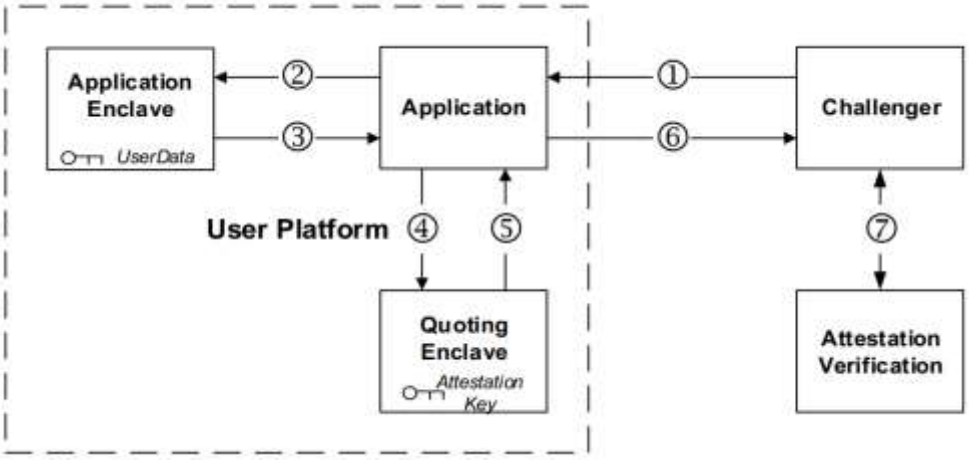
Target Enclave:

- Call EGETKEY to get its REPORT key
- Verifies MAC of REPORT from source enclave

SGX Remote Attestation

SGX uses a Quoting Enclave to convert LOCAL attestation to REMOTELY verifiable assertion (QUOTE)

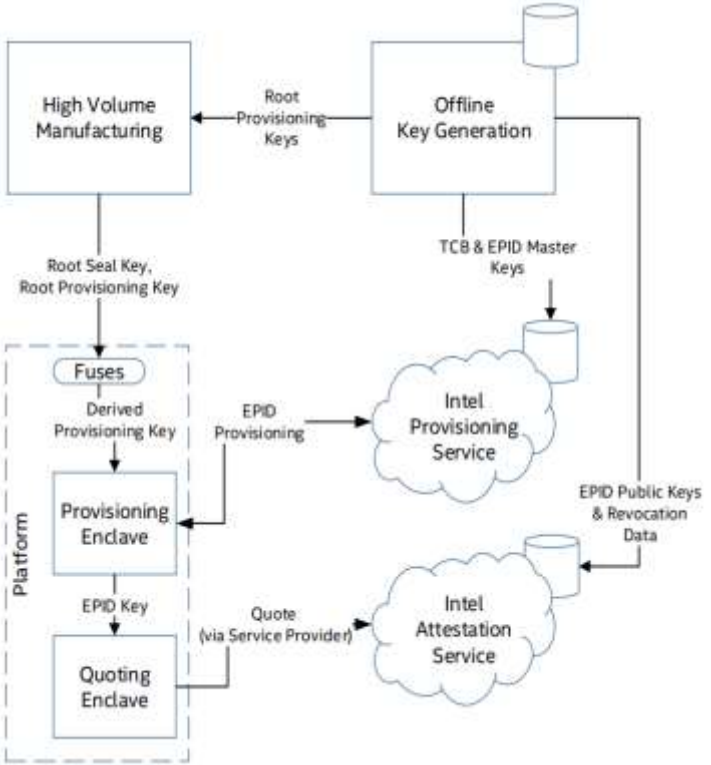
- Application Enclave (AE) generates EREPORT and gets verified by Quoting Enclave (QE) by local attestation.
- QE signs EREPORT w/ “Attestation Key” and converts into QUOTE, and sent to Challenger
- Challenger verifies QUOTE via “Attestation Key”.



Intel SGX Provisioning Service

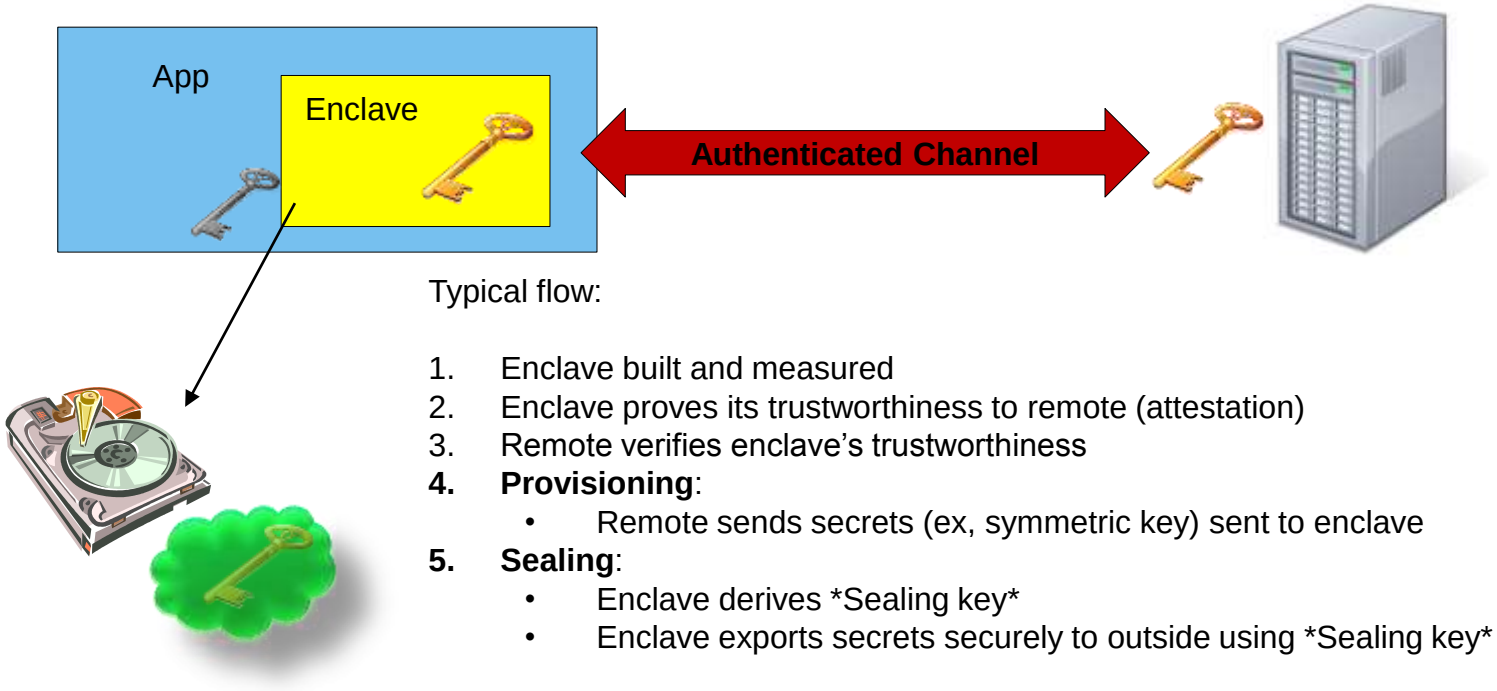
Attestation key (AK) must be established between Platform and Intel Attestation Service prior to remote attestation can be proceeded.

- Provisioning Enclave (PE) uses Provisioning key (PK) to demonstrate authenticity to Intel Provisioning Service (PS)
  - PK (fuse) is known to Intel
- PS provisions Attestation Key to PE after authenticity verified.
- PE sends Attestation Key to QE.
- Intel Attestation Services also knows Attestation Key therefore is able to attest.



Intel SGX infrastructure Services

SGX Sealing



Typical flow:

1. Enclave built and measured
2. Enclave proves its trustworthiness to remote (attestation)
3. Remote verifies enclave's trustworthiness
4. **Provisioning**:
   - Remote sends secrets (ex, symmetric key) sent to enclave
5. **Sealing**:
   - Enclave derives *Sealing key*
   - Enclave exports secrets securely to outside using *Sealing key*

SGX Sealing Key

- **Sealing Key**
  - Cryptographically protecting data when it leaves enclave
  - Can be based on different key derivation policies
- **Sealing Key Derivation**
  - **Enclave identity based** (selectable):
    - EGETKEY derives Sealing key based on MRENCLAVE
    - Ensure only specific enclave can imports secrets back to enclave
  - **Enclave Authority based** (selectable):
    - EGETKEY derives Sealing key based on MRSIGNER
    - Ensure only enclave from specific authority can imports secrets back to enclave
  - **Enclave version based** (always):
    - EGETKEY derives Sealing key based on KEYREQUEST.ISVSVN & KEYREQUEST.CPUSVN
      - Support secrets migration between old enclave and new enclave.

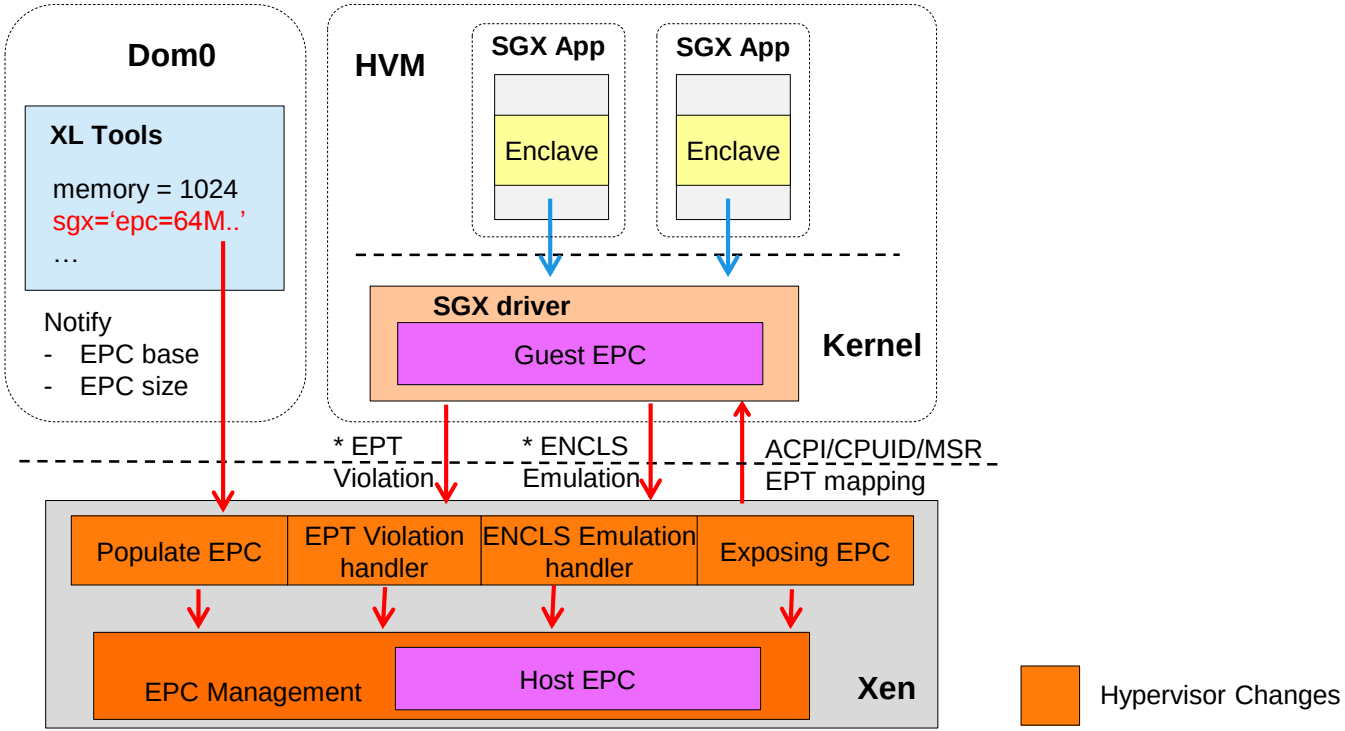
Agenda

- SGX Introduction
- **SGX Virtualization**
- Backup

Virtualizing SGX Overview

- General Enabling – Pretty Straightforward
  - SGX detection, EPC management
  - Expose virtual EPC to guest, w/ size configurable from user
  - SGX CPUID/MSR emulation
  - EPC Virtualization: How to allocate EPC to guest
  - Setup EPT mapping for guest EPC and host EPC.
  - SGX interaction with VMX (and others)
    - ENCLS VMEXIT handling, etc
- EPC Virtualization Approaches
  - Static Partitioning
  - Oversubscription
  - Ballooning

Xen SGX Virtualization Overview



* ENCLS emulation & EPT violation may not needed, depending on implementation of EPC virtualization

Query SGX (EPC) Info

- New XL commands to query physical EPC and domain EPC info (other SGX info?)
  - Xl sgxinfo: show physical EPC
  - Xl list <did>: show domain's EPC size
- Or extend existing XL commands
  - xl info -sgx
  - xl list <did> -sgx

New SGX XL Parameter to Create VM

- New parameter in XL configure: sgx = ‘epc=<size>,lehash=<sha256-hash>,lewr=<0|1>’
  - ‘epc’ specifies guest EPC size, which is configured by user. EPC base is calculated by toolstack (according to memory size, mmio configuration, etc).
  - ‘lehash’ specifies default MSR values; ‘lewr’ specifies whether to lock MSR in FEATURE_CONTROL.
- Behavior of ‘lehash’ and ‘lewr’
  - If physical machine is in *unlocked* mode
    - ‘lehash’ and ‘lewr’ behave as explained above; ‘lehash’ by default is Intel’s value; ‘lewr’ by default is 1.
  - If physical machine is in *locked* mode
    - Both cannot be specified; otherwise fail to create VM

EPC Management

	Pros	Cons
Integrate EPC to Memory Management framework	• Leverage MM code: page allocation algorithm, etc. • Flexible to support NUMA EPC (for SGX server machine).	Change to common MM code; Complicated to implement; -- high risk
Manage EPC separately	• Easy to implement (ex, EPC is limited, and can be organized in single-list) • No changes to common MM code – less risk	• Duplicated code; • Not flexible

EPC Virtualization -- Overview

- **Static Partitioning**
  - Allocate all EPC pages when VM is created
  - EPC allocated to KVM cannot be recycled by SGX driver
- **Oversubscription**
  - Allow creating multiple VMs with total virtual EPC bigger than physical EPC
  - EPC allocated to VM upon EPT violation.
  - Allow evicting & reloading EPC from/to VMs.
  - Much more complicated in implementation.
- **Ballooning**
  - Static Partitioning + EPC Ballooning support

VMM EPC Oversubscription

- **To support EPC oversubscription, VMM needs to know below from guests**
  - EPC page type (SECS, VA, regular, etc), status (blocked, etc)
  - SECS location
- **Without hardware feature support, VMM needs to trap ENCLS for above info**
  - Maintain all EPC & Enclave info from all VMs upon ENCLS VMEXIT.
  - VMM runs ENCLS on behalf of guest, and emulate result
    - VMM needs to reconstruct/remap ENCLS parameter (from guest VA to VMM VA).
- **w/ hardware feature support, easier but still complicated**
  - Use ERDINFO to get EPC info (EPC page type, status, GFN of SECS page, etc).
  - Use EINCVIRTCHILD/EDECVIRTCHILD to prevent SECS being evicted by guest
  - Restore GFN of SECS page when reload.
- **Maybe more importantly:**
  - Need algorithm to decide which EPC page to evict.

EPC Virtualization – Summery

	Pros	Cons
Static Partitioning	• Easy implementation (no ENCLS trapping/emulation, No EPT violation) • No hypervisor overhead	Potential inefficient use of EPC
Ballooning	• Pros of “static partitioning” • More efficient use of EPC	Require ballooning driver in guest
Oversubscription	More efficient use of EPC	• Complicated implementation • Higher hypervisor overhead

- We have preliminary patches on github with “static partitioning” implemented.
- Oversubscription and Ballooning?

CPUID & MSR Emulation

- CPUID Emulation
  - Expose native's SGX CPUID to guest, except
    - *virtual* EPC base and size are given by toolstack.
    - CPUID.0x12.0x1:XFRM needs to be consistent with guest's XCR0
  - Guest's *virtual* EPC base and size are notified to Xen via XEN_DOMCTL_set_cupid
  - EPC pages can be allocated and mapped to guest upon this.
- IA32_FEATURE_CONTROL MSR emulation
  - Always enable SGX in IA32_FEATURE_CONTROL, if SGX is exposed to guest.
  - LEWR bit is determined by 'lewr' from XL parameter (notified by XEN_DOMCTL_set_vcpu_msr)
- IA32_SGXLEPUBKEYHASH[0-3]
  - Hypervisor gets initial value by XEN_DOMCTL_set_vcpu_msrs
  - Keep per-vcpu variables, and update when MSR writes from guest
  - Write per-vcpu variable to physical MSR (for running LE in guest correctly)
    - Some optimizations can be done but need to trap EINIT.
- Several other MSRs: OwnerEpoch is written only, etc.

Live Migration, Checkpointing, Snapshot ?

- From HW's view, SGX cannot support live migration
  - SGX key architecture cannot be virtualized – they are bound to platform
  - EPC cannot be migrated
- However, with some facts, we can support live migration (sort of)
  - Losing platform keys are not problem, as they can be re-provisioned again.
    - Ex, Enclave author needs to handle losing sealing key anyway
  - Both Windows & Linux driver committed to support *sudden* lose of EPC
    - We can support live migration by just ignoring EPC.
    - Destination VM suffers *sudden* lose of EPC, which can be handled.
    - However this is not HW behavior (in HW, EPC only got lost in S3-S5, shutdown, reset).
- Checkpointing & Snapshot can also be supported by ignoring EPC.

ACPI – Example from Real Machine

```
Device (EPC)
    {
        Name (_HID, EisaId ("INT0E0C"))  // _HID: Hardware ID
        …
        Name (RBUF, ResourceTemplate ()
        {
            QWordMemory (ResourceConsumer, PosDecode, MinNotFixed, MaxNotFixed, NonCacheable, ReadWrite,
                0x0000000000000000, // Granularity
                0x0000000000000000, // Range Minimum
                0x0000000000000000, // Range Maximum
                0x0000000000000000, // Translation Offset
                0x0000000000000001, // Length
                ,, _Y56, AddressRangeMemory, TypeStatic)
        })
        Method (_CRS, 0, NotSerialized)  // _CRS: Current Resource Settings
        {
            CreateQWordField (RBUF, \_SB.EPC._Y56._MIN, EMIN)  // _MIN: Minimum Base Address
            CreateQWordField (RBUF, \_SB.EPC._Y56._MAX, EMAX)  // _MAX: Maximum Base Address
            CreateQWordField (RBUF, \_SB.EPC._Y56._LEN, ELEN)  // _LEN: Length
            EMIN = \_PR.EMNA /* External reference */
            ELEN = \_PR.ELNG /* External reference */
            EMAX = ((\_PR.EMNA + \_PR.ELNG) - One)
            Return (RBUF) /* \_SB_.EPC_.RBUF */
        }
        …
```

Exposing vEPC via ACPI

- Rational: Windows SGX driver requires ACPI table
- Current approach:
  - Hvmloader gets virtual EPC info by CPUID
  - vEPC info stored to ‘struct acpi_info’
  - New EPC node in DSDT table
  - In _CRS method, we setup EPC base & size by referring to ‘struct acpi_info’ in DSDT.

Questions?