Introduction to Intel Software Guard Extensions

Nico Weichbrodt / @zenvy

2019-07-14

Intel Software Guard Extensions (SGX)

- Extension of the x86 instruction set
- Introduced with the Skylake architecture
- Allows creation of *enclaves*
  - i.e., isolated compartments
- Enclaves are meant to process sensitive data securely

→ Small, tailored application parts can be enclavised

- All enclave memory is encrypted
- Enclave integrity is verified after creation by SGX
- Needed trust is reduced to Intel and the CPU package

Use Cases – Three Sides

1) I want to run my software on my system
   - Probably don't need SGX, I trust my system
   - Can still be used to protect secrets against malware
     - e.g., to hide keys

Use Cases – Three Sides

2) I want to run my software somewhere else securely

- I want to use a more powerful machine
- I don't trust the remote system / provider

- Computation without data disclosure
- Enclave contains sensitive data, like keys
  - e.g., SSH host keys, LUKS master key, ...
  - → All crypto has to go through enclave
  - or medical data / genetic data

→ *I* can do stuff without the provider seeing what I do

Use Cases – Three Sides

3) Someone else wants to run their software on my system

- A remote party does not trust me
- Proprietary software can generate and hide secrets from me
- Can be used to implement rights management
  - e.g. only able to start software x times
  - or use feature y only x times
  - or decrypt 4K BluRays and reencrypt them with HDCP

$\rightarrow$ *Someone else* can do stuff without me seeing what they do

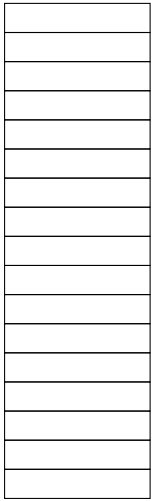
Enclave Overview

- Enclaves are part of normal applications
  - But even less privileged
  - Live in special memory region
- Only runnable in user space, no kernel enclaves
  - Creation only in kernel space
- No system calls allowed (fault on `int` and `syscall`)
- Some instructions are not allowed
  - e.g. `cpuid`
- No direct calls into the enclave allowed
  - Predefined entry points for entering
- No direct calls out of the enclave allowed
  - Special exit instruction, but destination not fixed

$\rightarrow$ Secure computation in cooperation with untrusted software

Enclave Page Cache (EPC)

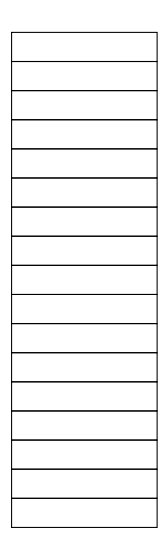
- Memory for all enclaves
- In current implementations part of system memory
  - Mapped as processor reserved memory (PRM)
- PRM size max 128 MiB (2^{15} = 32768 x 4 KiB pages)
  - Will increase in the future



Enclave Page Cache (EPC)

- PRM is always encrypted
- Memory Encryption Engine inside CPU
- Key generated on boot, kept in CPU
- Regenerated after sleep
  - States S3-S5 will drop key

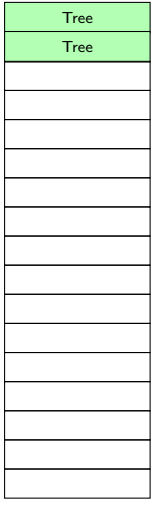
→ Enclaves do not survive hibernation and standby



Enclave Page Cache (EPC)

- Always integrity protected
- Hash tree
  - Root hash inside the CPU package
  - Tree inside the PRM
  - → Some EPC is lost
- Every access checks the tree

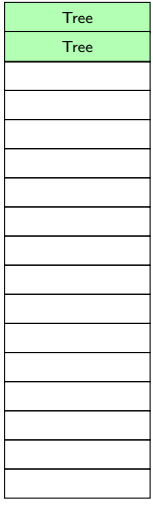
PRM / EPC



Enclave Page Cache (EPC)

- Swapping pages from EPC to memory is supported
- One EPC page is needed to hold version information of swapped out pages
- These can be swapped out, too
- High performance cost due to page faults

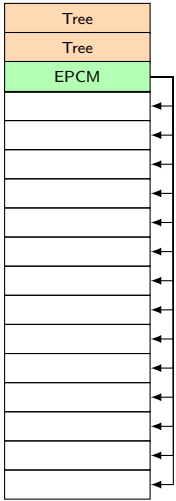
PRM / EPC



Enclave Page Cache Map (EPCM)

- Tracks state of all pages in EPC
- Is a “micro-architectural” data structure
  - Occupies some PRM
  - → 93 MiB left for enclaves
- Contains information like
  - Is page mapped?
  - Permissions
  - Page type
- Size of EPCM determines size of EPC

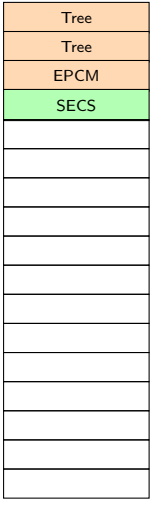
PRM / EPC



SGX Enclave Control Structure (SECS)

- One per enclave
- Contains most enclave metadata
  - Size, hash, ...
- Inaccessible from untrusted and trusted side
  - Only by the CPU itself
- Immutable after creation

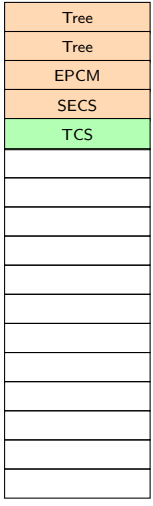
PRM / EPC



Thread Control Structure (TCS)

- Enclaves must have at least one TCS
- Describes an entry point into the enclave
- Multithreading is supported by SGX
- # of TCS = # of enclave threads
- Inaccessible from untrusted side
  - Enclave can read out some fields
- Immutable after creation
- References the State Save Area (SSA)

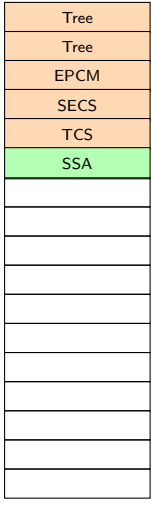
PRM / EPC



State Save Area (SSA)

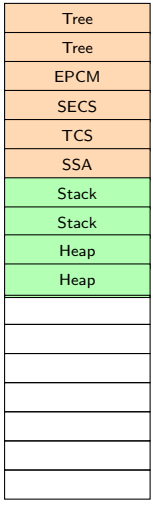
- Saves the processor state during interrupts
- At least on SSA per TCS needed
- Written on interrupt
- Read on resume

PRM / EPC



Enclave Stack and Heap

- Enclave *should* have its own stack and heap
- Not enforced by SGX
  - Recommended for obvious reasons
- `rsp` and `rbp` are saved on enter
  - But not changed!
  - Restored on exit
- Nothing is done for the heap
  - You need your own allocator

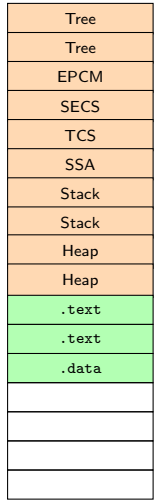


Code and Data Pages

- Your code and data goes here
- Not encrypted before creation
  - But integrity checked
- Enclave code and data is public until startup

→ Enclaves cannot have secrets at startup

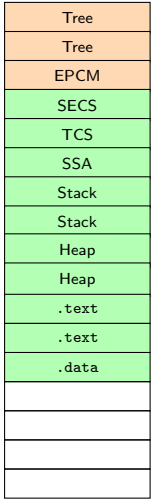
- Inject them later
- Remaining EPC can be used by other enclaves



Enclave itself

- Consists of multiple data structures
  - SECS - one per enclave
  - TCS - one per thread
  - SSA - multiple per thread
- + own stack
- + own heap
- + `.text`
- + `.data`

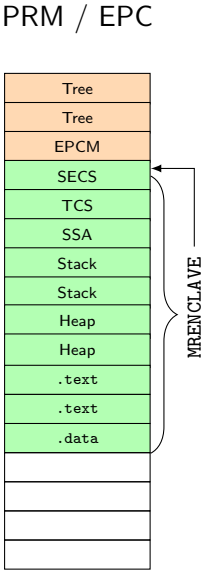
PRM / EPC



Enclave Measurement

Each enclave has its own unique hash sum comprised of its page layout and contents

- *Measurement* of the enclave
- Two enclaves with the same measurement are the same enclave
- Called `MRENCLAVE`
- Used for integrity protection



Instruction Overview

ENCLS	ECREATE	Starts the enclave creation process
	EADD	Adds pages to an enclave during creation
	EEXTEND	Calculates the hash sum of newly added pages
	EINIT	Finalize the creation process
ENCLU	EENTER	Enter an enclave
	EEXIT	Exit an enclave
	ERESUME	Resume an enclave

ENCLS kernel mode ENCLU user mode

Not a complete list

Enclave Development

- No special compiler
- Enclave must be self contained
  - Statically linked
  - No system calls
  - `-nostdinc -nostdlib -nodefaultlibs -nostartfiles`
- Launch needs a `SIGSTRUCT` and `EINITTOKEN`
  - Contains the `MRENCLAVE` and is signed
- `EINITTOKEN` is signed by an enclave trusted by the processor
  - e.g., Intel's launch enclave
  - New: Flexible Launch Control[1]
- Enter and exit through special instructions (`EENTER` and `EEXIT`)
- An SDK exists: `https://github.com/intel/linux-sgx`

[1] `https://github.com/intel/linux-sgx/blob/master/psw/ae/ref_le/ref_le.md`

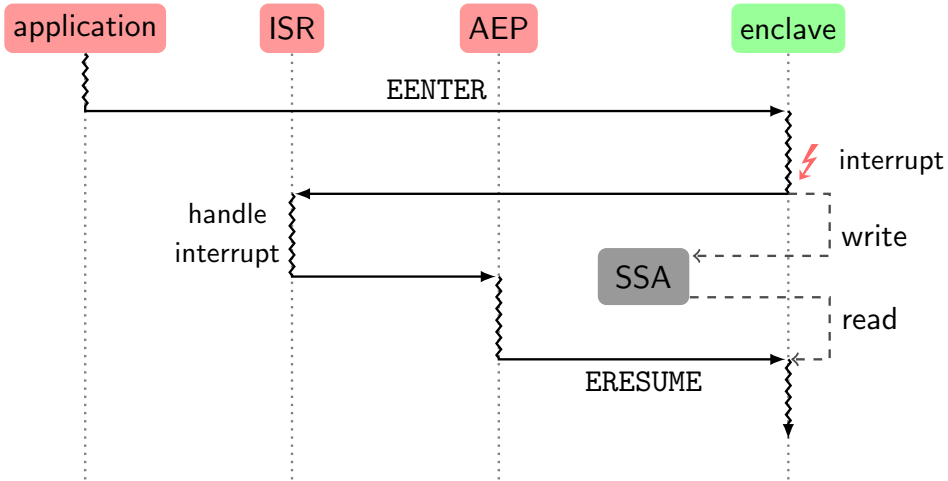
Enclave and Interrupts

Interrupts are transparent to enclaves

- They generate asynchronous enclave exits (AEX)
- On enter, an asynchronous exit handler (AEP) is registered
- AEP is called every time an AEX occurs
- AEP decides to resume enclave or not
  - SDK default: always resume
  - Not (easily) changeable

The same mechanism is used if an exception or fault occurs inside the enclave

Handling an AEX



Attestation

- Enclaves can prove to us, that they are enclaves
  - And that they are *the* enclave they claim to be
- This allows for secure communication with enclaves
- Local Attestation
  - Between two enclaves on the same system
- Remote Attestation
  - Between an enclave and a remote party

Local Attestation

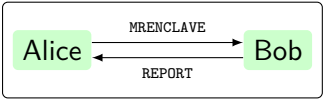
- `EREPORT` instruction for generating *reports*
  - HMAC'd by the processor
  - Made for a specific target enclave
- `EGETKEY` instruction to obtain the report key
- Reports can have a payload (64 Byte)
  - E.g. usable as a nonce for DH
  - Or hash of some public key

→ Secure channel between two enclaves

Local Attestation

Enclaves Alice and Bob want to verify their identity to each other

1. Alice sends her `MRENCLAVE` to Bob
2. Bob executes `EREPORT` with Alice's `MRENCLAVE`
3. Bob sends the report to Alice
4. Alice verifies the received report
   - Using a key obtained via `EGETKEY`
5. Repeat for other direction

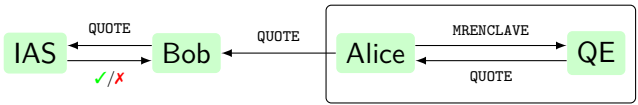


Remote Attestation

- Special enclave called Quoting Enclave (QE)
  - Signed by Intel
  - New: With Flexible Launch Control you can roll your own QE[2]
- Creates a *quote* from a report

1. Alice does local attestation with QE
2. Quote is sent to Bob (remote)
3. Bob asks the Intel Attestation Service (IAS) to verify quote

→ Secure channel between enclave and remote party



[2] https://github.com/intel/SGXDataCenterAttestationPrimitives

Sealing

- Enclave might want to store data persistently
- `EGETKEY` instruction to create a key based on either
  - `MRENCLAVE`
  - `MRSIGNER`
- Key also contains platform specific values

→ Same enclave on different platform get different keys

1. Get key
2. Encrypt data
3. Give it to untrusted system to store it

You cannot trust the untrusted system, yet you need it to actually store the data

Sealing

Sealing with an `MRENCLAVE` key

- Only this enclave can read the data

Sealing with an `MRSIGNER` key

- All enclaves signed by the same signing key can read the data
- Allows for forward compatibility and enclave updates
  - Enclave v2 can read v1 sealed data

Monotonic Counters

- SDK allows access to monotonic counters
- Preserve state after enclave destruction
- Managed by Intel ME
- Not available on every SGX-capable platform

Creating a monotonic counter (MC) involves writing to the non-volatile memory available in the platform.
Repeated write operations could cause the memory to wear out during the normal lifecycle of the platform.

Beware: An attacker can prevent your enclave from accessing a MC and force you to acquire a new one!

SDK

- Windows and Linux SDK available
  - For C/C++ development
  - Ships with an in-enclave `libstdc/libstdcxx`
  - Supports C++11
- Linux SDK is open source
  - Driver `https://github.com/intel/linux-sgx-driver`
  - SDK + PSW `https://github.com/intel/linux-sgx`
  - Binaries `https://01.org/intel-softwareguard-extensions`
- Windows SDK is not open source
  - But shares parts with Linux SDK
- SDK has an simulation mode, you don't need hardware
- Contains some samples

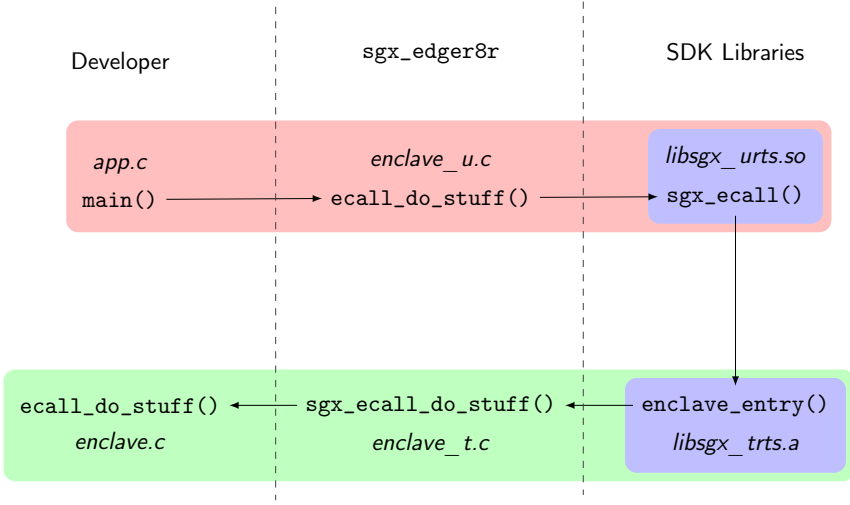
Enclave Interface

- SDK allows definition of `ECALLS` and `OCALLS`
  - `ECALLS` are transitions from untrusted to trusted code
  - `OCALLS` are the opposite

```
enclave {
  untrusted {
    void ocall_print_string ([in, string] const char *str);
  };
  trusted {
    public void ecall_do_something (int foo, [in, size=len] uint8_t *bar,
                                    size_t len);
  };
};
```

See Developer Reference for a detailed description

SDK Call Hierarchy



Alternative SDKs / Languages

- Fortanix Rust Enclave Development Platform (EDP)
  - `https://edp.fortanix.com/`
  - `https://github.com/fortanix/rust-sgx`
  - Own Rust target, own tools
- Baidu Rust SGX SDK
  - `https://github.com/baidu/rust-sgx-sdk`
  - Built on top of the Intel C/C++ SGX SDK
  - Uses EDL abstraction and ECall/OCall concept

Interpreted languages?

- JavaScript: `node-secureworker`
  - `https://github.com/luckychain/node-secureworker`
  - Duktape JS engine ported into enclave

Attacks against SGX itself

- Rollback attacks against swapped out pages ✗
  - EPC versions are tracked
- Rowhammer on the EPC ✗/✓
  - EPC pages are protected by hashes stored in EPCM
  - SGX-Bomb: Locking down the processor via Rowhammer attack
- Critical bugs in the SGX specification ?
  - Someone would need to check, spec is open
  - Intel is fairly confident they don't have any
- Critical bugs in the SGX implementation ?/✓
  - Depends if you count side channels
- Breaking into ME also breaks SGX ?/(✓)
  - Breaks monotonic counters and trusted time
  - We think ME is not able to read EPC

Attacks against the enclave application

- Rollback attacks against sealed data ?/✓
  - Can be mitigated with monotonic counters
  - But otherwise no protection except for enclave versions
- Iago and TOCTTOU attacks ?/✓
  - Depends on the implementation of the application

Attacks against the SDK

- TOCTTOU attacks ✗/?
  - SDK copies `ECALL` arguments, if desired
  - Does not deep-copy structures (no following pointers)
- SDK is open source, go find bugs!

Side Channels against SGX

- *Spectre Attacks: Exploiting Speculative Execution*
- *Foreshadow: Extracting the Keys to the Intel SGX Kingdom with Transient Out-of-Order Execution*
- *ZombieLoad: Cross-Privilege-Boundary Data Sampling*

All fixed by microcode updates (at cost of transition performance)
6k $\rightarrow$ 10k (w/ Spectre µCode) $\rightarrow$ 13k cycles (w/ Foreshadow µCode)
SDK also contains `mfence` to mitigate Spectre

All side channels are not exclusive to SGX!

Side Channels against Application

Depends on the implementation of the application

- *AsyncShock: Exploiting Synchronisation Bugs in Intel SGX Enclaves*
- *Controlled-Channel Attacks: Deterministic Side Channels for Untrusted Operating Systems*
- *Telling Your Secrets Without Page Faults: Stealthy Page Table-based Attacks on Enclaved Execution*
- *Inferring Fine-grained Control Flow Inside SGX Enclaves with Branch Shadowing*

Your application cannot trust anything, harden it as much as possible!

SGX v2 and SGX OVERSUB

SGXv1:

- Enclave layout is immutable after creation
- No adding pages, no changing permissions

→ People make heap executable to load code after enclave creation

SGXv2:

- New instructions to add/remove pages
- New instructions to change permissions
- Intel SDK has rudimentary support
- Only Gemini Lake CPUs support this

OVERSUB:

- New instuctions to virtualise EPC better
- No idea if this is available already

Hardware? Software?

Did you buy a new desktop/laptop machine in the last 3 years?

→ you probably have SGX

- BIOS/EFI has to turn it on

Software using SGX

- Cyberlink PowerDVD for 4K BluRay playback
- Prototype for Signal Contact Discovery[3]
- Fortanix Self-Defending Key Management Service[4]
- ⟨your software here⟩

[3] `https://signal.org/blog/private-contact-discovery/`
[4] `https://fortanix.com/products/sdkms/`

Summary

- SGX enables trusted execution via enclaves
- Memory encryption, integrity checks
- SDK(s) is/are available
- You can develop software using it, today
- Attacker model changes drastically

Summary

- SGX enables trusted execution via enclaves
- Memory encryption, integrity checks
- SDK(s) is/are available
- You can develop software using it, today
- Attacker model changes drastically

Questions?

Sources

- Intel 64 and IA-32 Architectures Software Developer Manual
  - aka THE manual, SGX start at page 4177
  - Volume 3, chapters 36-42
  - `https://software.intel.com/en-us/articles/intel-sdm`
- The papers mentioned in the talk
  - find them on Google Scholar or talk to me