Introduction to Intel Software Guard Extensions

Nico Weichbrodt @zenvy

HoA HoA HoA Täterä

2018-08-04

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 Software Guard Extensions (SGX)
- Needed trust is reduced to Intel and the CPU package

Use Cases – Three Sides

1) I want to run software on my system

- Probably don't need SGX, I trust my system
- Can still be used to protect secrets against malware

Use Cases – Three Sides

2) I want to run 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
  - SSH host keys, luks master key, ...

  → All crypto has to go through enclave

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

Use Cases – Three Sides

3) Someone else wants to run 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

→ *Someone* can do stuff without me seeing what they do

Enclave Overview

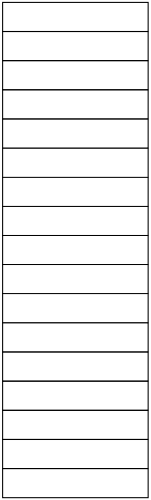
- Enclaves are part of normal applications
  - But even less privileged
- Only runnable in user space, no kernel enclaves
  - Creation only in kernel space
- No system calls allowed (fault on `int` and `syscall`)
- Some instructions are not allowed
  - e.g. `cpuid`
- No direct calls into the enclave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

→ 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
- Max 128 MiB ($2^{15} = 32768$ x 4 KiB pages)

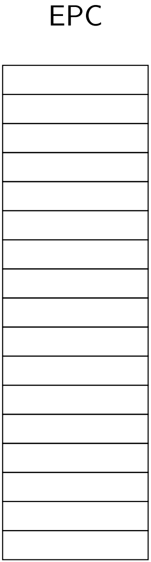
EPC



Enclave Page Cache (EPC)

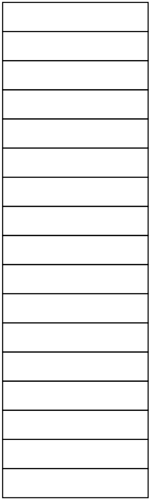
- EPC is always encrypted
- Memory Encryption Engine inside CPU
- Key generated on boot, kept in CPU
- Regenerated after sleep

→ Enclaves do not survive hibernation and standby



Enclave Page Cache (EPC)

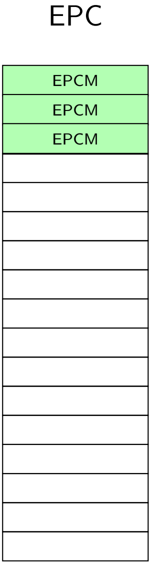
EPC



- 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

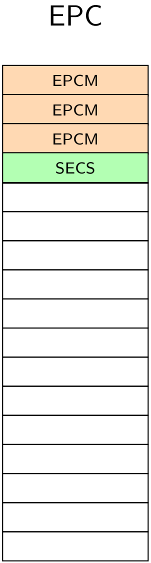
Enclave Page Cache Map (EPCM)

- Tracks state of all pages in EPC
- Is a "micro-architectural" data structure
  - Occupies 35 MiB of EPC
  - → 93 MiB left for enclaves
- Contains information like
  - Is page mapped?
  - Permissions
  - Page type
- Size of EPCM determines size of 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

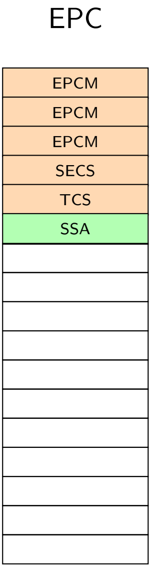


Thread Control Structure (TCS)

- 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)

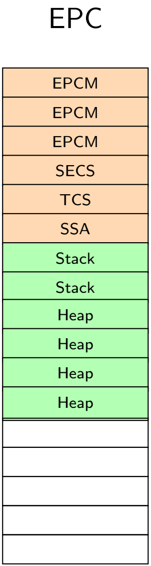
State Save Area (SSA)

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



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

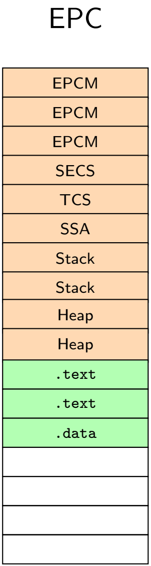


Code and Data Pages

- 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

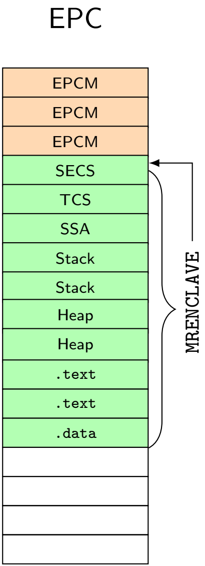
- Inject them later
- Remaining EPC can be used by other enclaves



Enclave 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
- Often called `MRENCLAVE`
- Used for integrity protection



Enclave Development

- No special compiler
- Enclave must be self contained
  - Statically linked
  - No system calls
  - `-nostdinc -nostdlib -nodefaultlibs -nostartfiles`
- Developer must provide 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
- Enter and exit through special instructions
- An SDK exists: `https://github.com/intel/linux-sgx`

Instruction Overview

`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
`EENTER`	Enter an enclave
`EEXIT`	Exit an enclave
`ERESUME`	Resume an enclave

kernel mode user mode

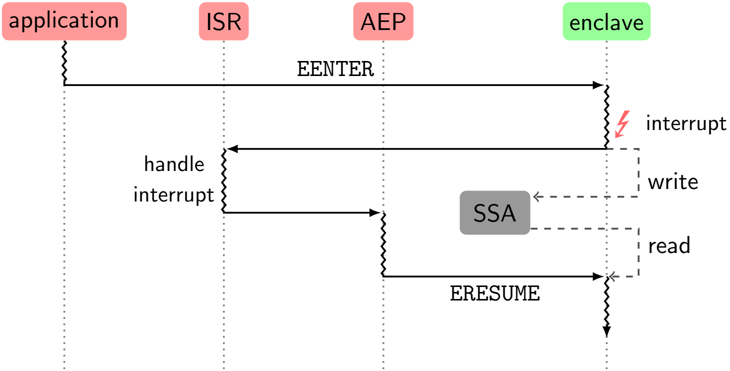
Enclave and Interrupts

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

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

- Alice send her `MRENCLAVE` to Bob
- Bob calls `EREPORT` with Alice's `MRENCLAVE`
- Bob sends the report to Alice
- Alice verifies the received report
  - Using a key obtained vie `EGETKEY`
- Repeat for other direction

Remote Attestation

- Special enclave called Quoting Enclave (QE)
  - Signed by Intel
- Creates a *quote* from a report

1. Enclave does local attestation with QE
2. Quote is sent to remote party
3. Remote party asks Intel to verify quote

→ Secure channel between enclave and remote party

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

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/01org/linux-sgx-driver`
  - SDK + PSW `https://github.com/01org/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

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 on 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

- Depends on the implementation of the application
- Spectre
- *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*

Hardware? Software?

Did you buy a new desktop/laptop machine in the last 2 years?

→ you probably have SGX

Software using SGX

- Cyberlink PowerDVD for 4K BluRay playback
- Prototype for Signal Contact Discovery[1]
- ⟨your software here⟩

[1] https://signal.org/blog/private-contact-discovery/

Sources

- Intel 64 and IA-32 Architectures Software Developer Manual
  - aka THE manual, SGX start at page 4103
  - `https://software.intel.com/en-us/articles/intel-sdm`
- The papers mentioned in the talk
  - find them on Google Scholar or talk to me