White Paper

Asynchronous Enclave Exit Notify and the EDECCSSA User Leaf Function

Table of Contents


```
IF (TCS.FLAGS.AEXNOTIFY = 1 &&
    SSA[TCS.CSSA-1].GPRSGX
                   .AEXNOTIFY[0] = 1)
Then
   << Do EENTER flow >>
Else
   << Do original ERESUME flow >>
FI;
```

Listing 1: ERESUME pseudocode flow

```
If (TCS.CSSA == 0) #GP(0);
<< Validate TCS.SSA[TCS.CSSA-1] >>
TCS.CSSA <- TCS.CSSA - 1;
```

Listing 2: EDECCSSA pseudocode flow

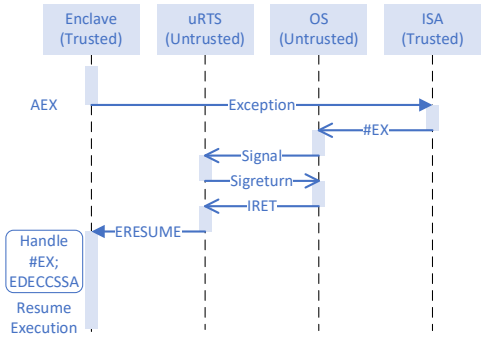

Figure 1: Enclave exception handling flow

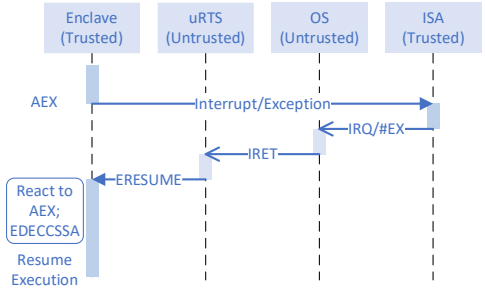

Figure 2: Enclave AEX notification flow

1. Introduction and Overview

Asynchronous Enclave Exit Notify (AEX-Notify) is an architectural extension to Intel® SGX that allows SGX enclaves to be notified after an asynchronous enclave exit (AEX) has occurred. This mechanism can be used by enclave software to react to interrupts and exceptions by, for example, applying a mitigation. EDECCSSA is a new Intel SGX user leaf function that can facilitate AEX notification handling, as well as software exception handling. This white paper explains the modifications to the Intel SGX architecture that support AEX-Notify and the EDECCSSA user leaf function.

The following list summarizes the additions to existing Intel SGX data structures to support AEX-Notify (further details are provided in Section 6):

- **SECS.ATTRIBUTES.AEXNOTIFY:** This enclave supports AEX-Notify
- **TCS.FLAGS.AEXNOTIFY:** This enclave thread may receive AEX notifications
- **SSA.GPRSGX.AEXNOTIFY:** Enclave-writable byte that enables AEX notifications

A thread can only enter an enclave through a Thread Control Structure (TCS) if SECS.ATTRIBUTES.AEXNOTIFY = TCS.FLAGS.AEXNOTIFY. Listing 1 summarizes the behavior of ERESUME. An AEX notification will be delivered to the enclave thread if:

1. The enclave thread allows AEX notifications (TCS.FLAGS.AEXNOTIFY is set),
2. SSA[TCS.CSSA-1].GPRSGX.AEXNOTIFY[0] is set for the TCS being resumed,
3. and TCS.CSSA (the current SSA pointer) is greater than zero.

Note that AEX increments TCS.CSSA, and ENCLU[ERESUME] decrements TCS.CSSA, except when an AEX notification is delivered. Instead of decrementing TCS.CSSA and restoring state from the State Save Area (SSA), ENCLU[ERESUME] delivers an AEX notification by behaving as ENCLU[EENTER]. Implications of this behavior include:

1. The enclave thread is resumed at EnclaveBase + TCS.OENTRY
2. The enclave thread receives the (non-decremented) value of TCS.CSSA in EAX
3. RCX contains the address of the IP following ENCLU[ERESUME]
4. The state saved by the most recent AEX is preserved in SSA[TCS.CSSA-1]

The enclave can return to the previous SSA context by invoking an ENCLU leaf called EDECCSSA, shown in Listing 2. Before invoking ENCLU[EDECCSSA], enclave software should preserve all required state from SSA[TCS.CSSA-1], for example, GPRs and XSAVE state. After invoking ENCLU[EDECCSSA], the enclave may jump to the RIP where the most recent AEX had occurred. Section 3 describes an example software model that uses AEX-Notify and the EDECCSSA leaf function.

Note that if the OS/VMM does not want to allow an enclave to receive AEX notifications, it can clear SECS.ATTRIBUTES.AEXNOTIFY when creating the enclave.

2. Changes to Enclave Flows

Figure 1 illustrates how AEX-Notify can allow enclaves to handle exceptions more efficiently. When an exception triggers an AEX, enclave execution is suspended, and control is transferred to the OS. The OS may then choose to deliver an exception signal to the uRTS (untrusted runtime system), which in turn can decide whether to allow the enclave to handle the exception; if so, the uRTS will unwind the exception by issuing a sigreturn and then returning to the enclave. Immediately following ENCLU[ERESUME], the enclave is notified that an AEX had occurred, and it can respond by handling the exception. The enclave thread may then resume execution where the AEX had occurred.

Figure 2 shows the case where the enclave is not expected to handle an exception but may still want to be notified that an AEX had occurred, for example, to keep a count of the number of AEXs that have occurred during enclave execution.

3. Example Software Model

The majority of commodity SGX SDKs and frameworks implement a two-stage exception handler (with NSSA=2). The stage-1 handler runs in the SSA[1] context, and the stage-2 handler runs in the SSA[0] context. The enclave main flow also runs in the SSA[0] context. This existing software exception handling flow can be extended to handle AEX Notifications. The required modifications may include:

- Using ENCLU[EDECCSSA] (instead of ENCLU[EEXIT] followed by ENCLU[ERESUME]) to transition to the stage-2 handler
- Extending the stage-2 handler to react to the AEX
- Managing the SSA.GPRSGX.AEXNOTIFY bit(s) to enable AEX notifications only when they are needed. For example, the enclave may not want to be notified if an interrupt occurs while the enclave is handling an exception

The following describes how an enclave thread can enable AEX-Notify, handle an AEX notification, and then return to the point at which the AEX had occurred:

1. The enclave main flow (in the SSA[0] context) enables AEX-Notify by setting bit 0 within SSA[0].GPRSGX.AEXNOTIFY
2. The enclave encounters an interrupt or exception, for example, at RIP=0x1000
3. An AEX is triggered, which increments TCS.CSSA
4. The enclosing application executes ENCLU[ERESUME] to continue enclave execution. With AEX-Notify enabled, ENCLU[ERESUME] behaves as ENCLU[EENTER] and therefore does not decrement TCS.CSSA
5. The enclave entry point notices that TCS.CSSA=1 and jumps to the stage-1 handler (possibly after some other checks)
6. Stage-1 handler flow:
    a. Expand the stack if the remaining stack space is lower than a specified threshold
    b. If the checkpoint was set in step 7.b but was not cleared in step 7.f (thus indicating that an AEX occurred between these steps):
        i. Unwind the stack to the checkpoint created in step 7.b (for example, by restoring RSP from the TLS variable)
        ii. Execute ENCLU[EDECCSSA]
        iii. Jump to step 7.c
    c. Copy the SSA[0] context to stack
        i. Copy SSA[0].GRPSGX to stack
        ii. SSA[0].XSAVE can be either (1) copied to stack, or (2) loaded back to processor registers if not used by the stage-2 handler
        iii. SSA[0].MISC should also be copied if it will be used by exception handlers
    d. Disable AEX-Notify in SSA[0] by clearing bit 0 within SSA[0].GPRSGX.AEXNOTIFY
    e. Execute ENCLU[EDECCSSA]
7. Stage-2 handler flow:
    a. Check for and handle an exception, if necessary
    b. Create a checkpoint (for example, by saving the current RSP to a TLS variable)
    c. Enable AEX-Notify by setting bit 0 within SSA[0].GPRSGX.AEXNOTIFY
    d. React to the AEX, if necessary
    e. Restore GPR (and/or XSAVE) context
    f. Invalidate the checkpoint created in step 7.b to indicate that an AEX has been handled the main flow is being resumed
    g. Jump back to main flow, for example, to RIP=0x1000.

The checkpoint mechanism is conceptually similar to a long jump, a well-known design pattern in the C standard library. Step 7.b is analogous to `setjump()`, and steps 6.b.i through 6.b.iii are analogous to `longjump()`. The checkpoint (also known as a jump buffer) can be kept in enclave thread-local storage (TLS), for example.

4. Signing, Attesting, and Sealing

This section provides example software models for signing, attesting, and sealing with enclaves that use AEX-Notify.

An enclave thread cannot receive AEX notifications unless its TCS.FLAGS.AEXNOTIFY bit is set to 1, and SECS.ATTRIBUTES.AEXNOTIFY is also set to 1. The conjunction of these two enabling bits is required to maintain compatibility with other platform features. It also has implications for enclave signing, attestation, and sealing, because TCS.FLAGS is typically included in the MRENCLAVE measurement.

- **Signing:** The enclave owner may want to be able to deploy a single enclave binary signed with a single SIGSTRUCT on hardware platforms that do support AEX-Notify, and on hardware platforms that do not support AEX-Notify. As the TCS.FLAGS.AEXNOTIFY enabling bit affects MRENCLAVE, this deployment model requires some additional software support. For example, the enclave owner may take a single measurement of the enclave that includes two sets of TCSs: one set with the TCS.FLAGS.AEXNOTIFY bit set, the other set with the bit cleared. Each platform can then build and measure the enclave with both sets of TCSs. Platforms that support AEX-Notify would enter the enclave through the first set; platforms that do not support AEX-Notify would only be able to enter through the second set because ENCLU[ERESUME] and ENCLU[EENTER] will issue a #GP if a reserved bit is set in TCS.FLAGS. As soon as the enclave has been entered once through either the first set or the second set, enclave software can

prevent any subsequent entry through TCSs within the other set. For instance, on the first entry the enclave can set an internal global flag to 1 if an AEX-Notify-enabled TCS was used, and 0 otherwise. Subsequent enclave entries can be gated by this flag.

- **Attestation:** Both enabling bits are exposed by attestation: TCS.FLAGS.AEXNOTIFY is included in the REPORT.MRENCLAVE measurement, and the SECS.ATTRIBUTES.AEXNOTIFY bit is revealed in REPORT.ATTRIBUTES.
- **Sealing:** When an enclave intends to seal sensitive data persistently, it uses the ENCLU[EGETKEY] instruction to retrieve an encryption key. This key is derived containing the components of the enclave's identity and a value provided by the enclave called KEYID. By providing different values of KEYID, the enclave can access multiple seal keys, for example to ensure that no key is overused. In the signing model described above, the enclave would include an indicator in KEYID to derive a different key, depending on whether AEX-Notify is enabled or disabled.

5. Enumeration

CPUID.(EAX=12H, ECX=0):EAX[11] indicates that the EDECCSSA user leaf function is supported.

CPUID.(EAX=12H, ECX=1):EAX[10] indicates that software can set SECS.ATTRIBUTES.AEXNOTIFY. The bit position 10 corresponds to the bit position of the AEXNOTIFY enclave attribute (see Table 1).

On some platforms, AEX-Notify and the EDECCSSA leaf function can be deployed in a microcode update (MCU), in which case the enumeration will change when the MCU is loaded. Some platforms support reverting MCUs, rather than committing them. Reverting the MCU that deploys AEX-Notify and the EDECCSSA user leaf function will remove support for these features, and their enumeration will change accordingly. This behavior has the following implications for platform software:

- After loading an MCU, the OS/VMM may query CPUID to determine whether AEX-Notify and the EDECCSSA user leaf function have become available.
- After reverting an MCU, the OS/VMM may query CPUID to determine whether AEX-Notify and the EDECCSSA user leaf function have become unavailable.
- If a VMM loads the MCU that deploys AEX-Notify and the EDECCSSA user leaf function, the VMM should only expose CPUID.(EAX=12H, ECX=0):EAX[11]=1 and CPUID.(EAX=12H, ECX=1):EAX[10]=1 to guest VMs if the VMM will not revert (or has already committed) the MCU.
- Prior to reverting the MCU that deploys AEX-Notify and the EDECCSSA user leaf function, the OS should clear EPC memory. If EPC memory is not cleared before reverting this MCU, then either of the following may occur after the MCU has been reverted:
  - Any enclave thread that attempts to execute ENCLU[EDECCSSA] will signal #GP(0).
  - Any thread that attempts to enter an enclave through a TCS with TCS.FLAGS.AEXNOTIFY set to 1 will signal #GP(0).

  Therefore, if the OS/VMM does not clear EPC memory before reverting an MCU, then existing enclaves may become inoperable.

6. Modifications to SGX Structures

Field	Bit position	Description
AEXNOTIFY	10	AEX-Notify feature enabled

Table 1: Changes to SECS.ATTRIBUTES

When CPUID.(EAX=12H, ECX=1):EAX[10] is 0, SECS.ATTRIBUTES.AEXNOTIFY is reserved and must be 0.

Field	Bit position	Description
AEXNOTIFY	1	AEX-Notify feature enabled (for this thread)

Table 2: Changes to TCS.FLAGS

Note that ENCLU[EENTER] and ENCLU[ERESUME] will issue a #GP when invoked on a TCS whose TCS.FLAGS.AEXNOTIFY does not match the enclave's SECS.ATTRIBUTES.AEXNOTIFY. This check is required for feature compatibility.

Field	Offset (Bytes)	Size (Bytes)	Description
AEXNOTIFY	167	1	Bit 0: Flag enabling AEX-Notify for this SSA context Bits 1-7: Reserved

Table 3: Changes to SSA.GPRSGX

SSA.GPRSGX.AEXNOTIFY[0] is set and cleared by enclave software.

7. Modifications to SGX Instructions

Instr. Leaf	EAX	RBX	RCX	RDX
EDECCSSA	09H (In)	Unused	Unused	Unused

Table 4: Changes to ENCLU

The EDECCSSA user leaf function verifies SSA[TCS.CSSA-1] and then decrements TCS.CSSA. This operation allows enclave software to decrement TCS.CSSA without exiting and re-entering the enclave.

8. Enclave Code Debug and Profiling

Whenever an opt-in enclave entry is used to perform enclave code debugging or profiling, the debugger or profiling tool may clear TCS.FLAGS.AEXNOTIFY to prevent AEX notifications from being delivered at each interrupt, breakpoint, trap, or other exception.

Sometimes it is necessary for an opt-in enclave entry to receive AEX notifications. For example, this may be required to debug enclave code that is responsible for handling an AEX notification. In this scenario the debugger can set TCS.FLAGS.AEXNOTIFY, but it cannot use RFLAGS.TF (or EFLAGS.TF) to single step through the enclave while TCS.FLAGS.AEXNOTIFY is set to 1, since each trap will cause a new AEX notification to be delivered upon the next ENCLU[ERESUME]. Instead, the debugger can use one of the following methods:

- Set TCS.FLAGS.AEXNOTIFY and RFLAGS.TF, and then invoke ENCLU[ERESUME] to deliver an AEX Notification. The enclave will exit after executing the first instruction at EnclaveBase + TCS.OENTRY. The debugger can then clear TCS.FLAGS.AEXNOTIFY and use TF to single step through the enclave code.
- Set TCS.FLAGS.AEXNOTIFY (and do not clear it) and use INT3 to step through the enclave. For example, after executing the N^{th} instruction and trapping at the $N+1^{st}$ instruction, the debugger can move INT3 (the breakpoint) from the $N+1^{st}$ instruction to the $N+2^{nd}$ instruction that will be executed.

9. Interaction with Intel Control-Flow Enforcement Technology (CET)

Intel® CET provides the following capabilities to defend against ROP/COP/JOP style control-flow subversion attacks:

- Shadow stack: Return address protection to defend against ROP.
- Indirect branch tracking (IBT): Free branch protection to defend against COP/JOP.

Each of these capabilities can be enabled for an Intel SGX enclave at creation. Note that once enabled, they remain enabled throughout the enclave's lifespan, including during any invocations of the two-stage exception handler. When either of these features is enabled (or when both are enabled) for the enclave, the handler should ensure that their state is preserved. Specifically, when the handler returns to the main flow, the features' state should be identical to what it was when the AEX was triggered in the main flow. The handler can preserve this state as follows:

- The CET state saved in CETSSA[TCS.CSSA-1] can be copied (for example, to the stack) in the stage-1 handler before invoking ENCLU[EDECCSSA]. Like SSA frames, CETSSA frames are also organized as an array indexed by TCS.CSSA. However, unlike SSA frames, CETSSA frames are protected by read-only page permissions which can help to preserve the integrity of saved CET state. The CET state is not protected by read-only page permissions while it is kept temporarily on the stack by the handler, so the handler should be carefully implemented to not overwrite or corrupt this temporary state while it is on the stack.
- An indirect JMP can be used instead of RET to return from the exception handler to the main flow. In most cases, RET cannot be used because the shadow stack (if enabled) would not have a matching return address, and because the IBT state (if enabled) can only be restored with an indirect JMP. The IBT state saved on the CETSSA contains a TRACKER bit that can be restored by invoking the indirect JMP instruction with the no-track prefix (to set TRACKER=0) or without the no-track prefix (to set TRACKER=1).
  - An implementation could, for example, have the indirect JMP instruction read the target address from thread-local storage. This would allow the stage-2 handler to return to the main flow without clobbering a general-purpose register.
- If the exception handler does not maintain a balanced call stack (for example, if it invokes more calls than returns) then it can manually restore the SSP (for example, by using the INCSSP instruction).

Note that the guidelines in this section apply to any enclave that uses Intel CET and implements a second-stage exception handler in the SSA[0] context, regardless of whether the enclave also uses AEX-Notify or the EDECCSSA user leaf function.