

Trusted Execution Environments

Chapter 4

- [1] J. Szefer, “Principles of secure processor architecture design,” *Synth. Lect. Comput. Archit.*, vol. 13, no. 3, pp. 1–173, 2018.

Protecting Software Within Trusted Execution Environment (TEE)

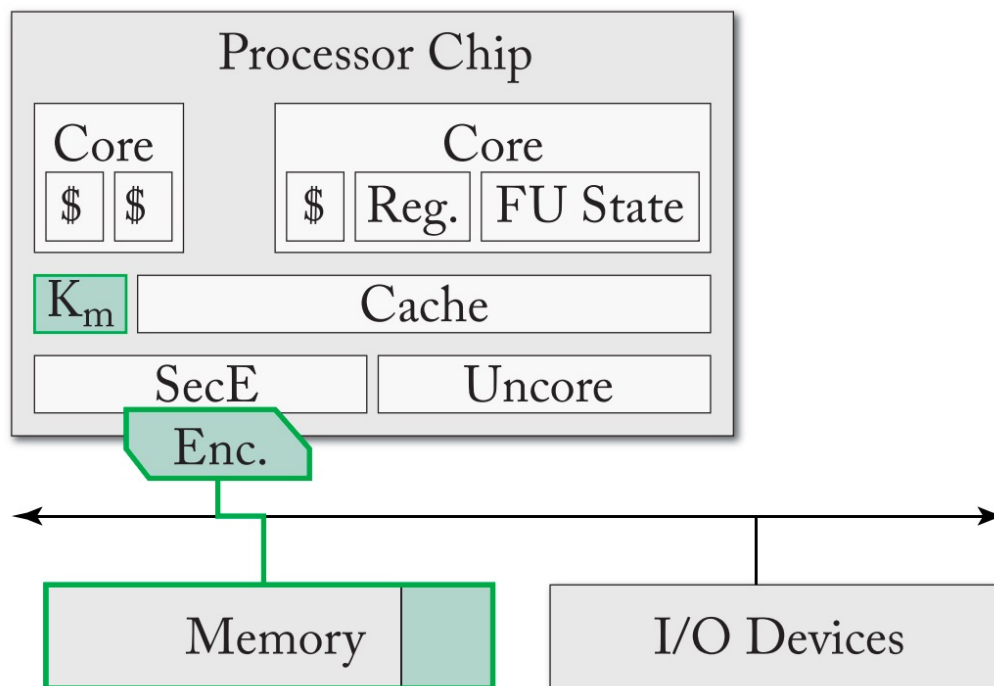
- ▣ TCB is responsible for creating TEE
- ▣ Software within TEE is protected against a range of software and hardware attacks (range is a function of threats model of SP, but **TEE is executed in the GPP** not the SP!!))
- ▣ Approximations
 - ◆ Protect Trusted Software Modules (TSM) or Enclaves
 - ◆ Protect VMs or containers
- ▣ All software within TEE is given the same set of protections (apart from the privilege levels differentiation in VMs)
- ▣ Users should be carefully about what code runs inside the TEE, especially **external libs**

Protections offered by the TCB to the TEE

- ▣ **TCB** provides **confidentiality** and **integrity** to the **TEE**
 - ◆ From potential attacks by other sw and hw outside (the TEE)
- ▣ No protection against **malicious/broken** TCB or **malicious/broken** TEE
- ▣ Multiple TEE/Enclaves running simultaneously is possible (e.g., from different users): TCB should prevent cross TEE attacks
- ▣ Most designs only consider **software-on-software** vector attacks
 - ◆ Hardware-on-hardware are partly considered (e.g., coldblood). Other cases not (e.g., hardware trojans)
 - ◆ Timing based side-channel is a **weak point** (it's a form of software-on-hardware vector attack)

Enforcing **Confidentiality** Through Encryption (I)

- Off-chip access untrusted by default → Hardware cyphering of the memory content (with an ephemeral key)

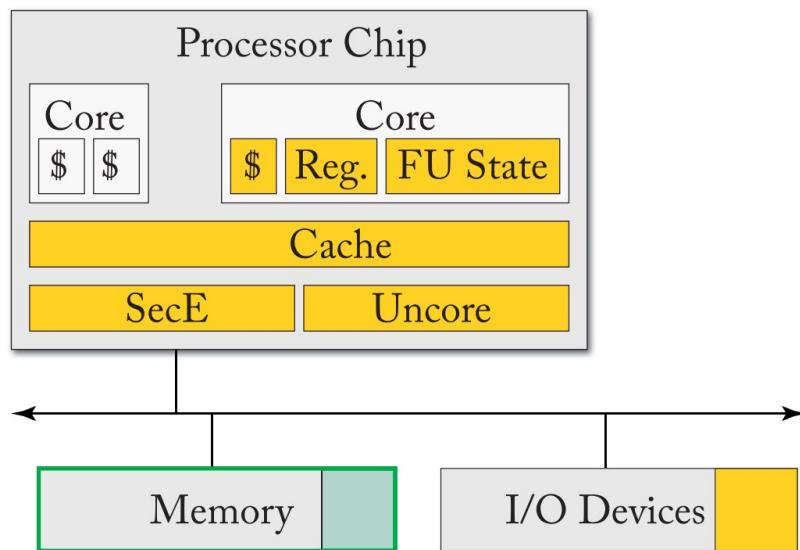


Enforcing **Confidentiality** Through Isolation (II)

- ▣ Page tables or Extended Page tables primary objective is isolation
- ▣ But, if Hypervisor/OS is untrusted the mechanism is not trusted
- ▣ If TCB should enforce isolation additional mechanism should be provided (e.g., Adding another level of translation) or architected as **dedicated memory management in TCB** that replace the table page-based mechanism
 - ◆ E.g., HyperWall[209] uses **hardware** (out of control of the hypervisor) to handle page tables
- ▣ No protection for hardware-on-hardware

Enforcing **Confidentially** Through State Flush (III)

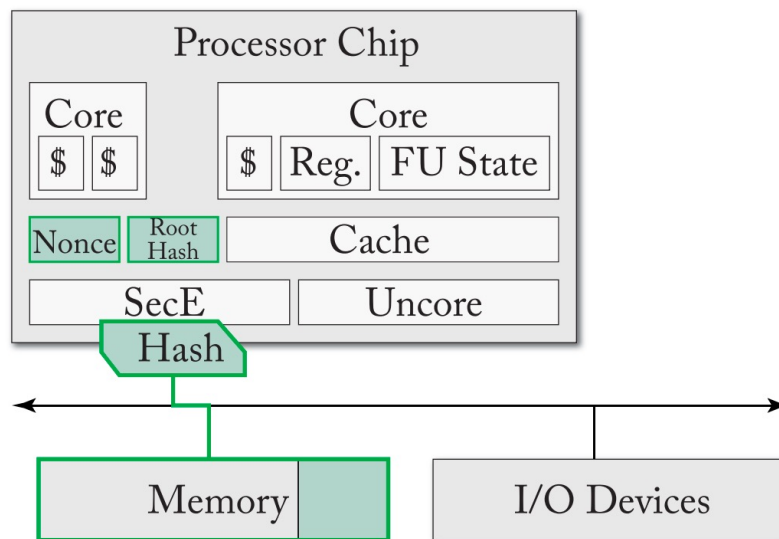
- Non architectural-state should be flushed once used by TEE
- Any register or execution dependent piece of information of the TEE should be deleted once the untrusted software continues after TEE
 - ◆ Speculative engines, cache contains, I/O traces,



Yellow == has to be flushed

Enforcing **Integrity** Through Cryptographic Hashing

- ▣ Add integrity to data going off-chip
- ▣ Event with data encrypted, some-one can change system behavior with out decrypting the data (e.g., replay attacks)
 - ◆ Add a nonce to prevent it.



Examples of Architectures for Protecting Enclaves

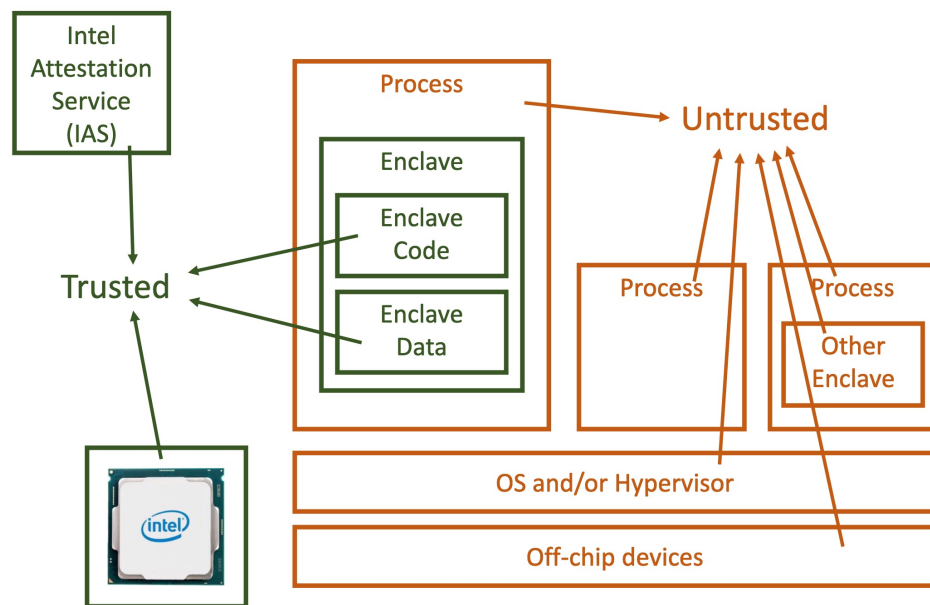
- Cell Broadband Engine and Processor Vault
 - ◆ Reserve a Synergistic Processing Element (SPE) for TEE.
 - ◆ SPE uses dedicated memory (is not shared across SPEs by design)
 - ◆ Uses public-key cryptography to be sent to processor vault and execute only signed code

- ARM TrustZone
 - ◆ Two separate worlds to execute trusted (secure OS) and untrusted (normal OS)
 - ◆ Memory and buses are tagged, allowing that some parts of the SoC are available to secure OS

- Intel SGX
 - ◆ Protection for trusted secure modules (called Enclaves)
 - ◆ Off-chip memory is protected via encryption and hashing
 - ◆ *Was weak against side-channel (encryption keys can be accessed)*
 - ◆ *Now can be protected via Resource Director (e.g., cache partitioning)*

Example: Intel SGX (Hardware Enclave)

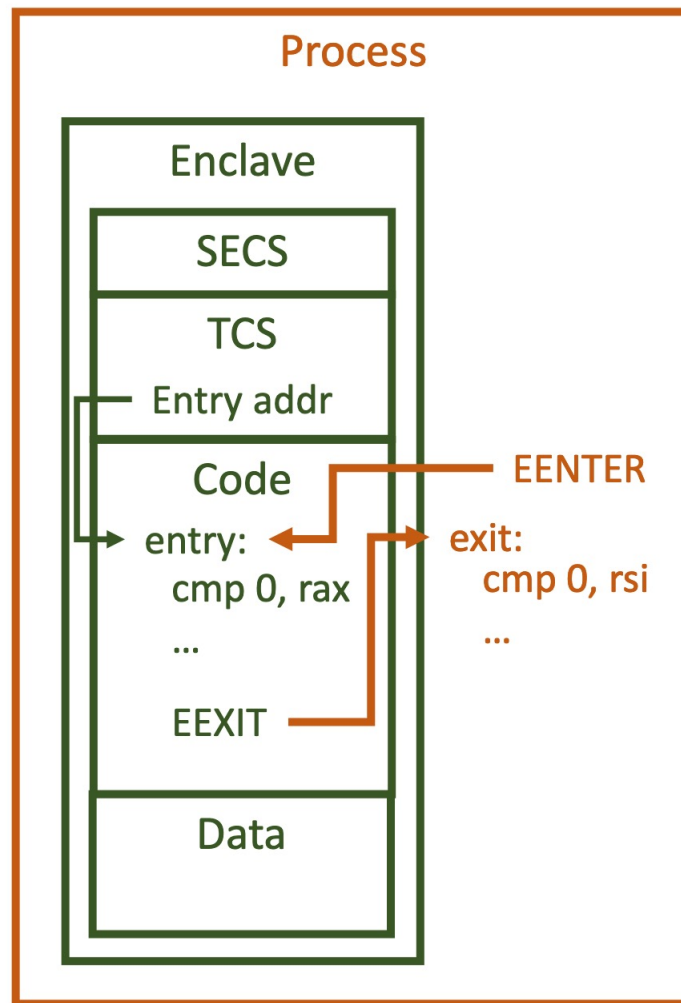
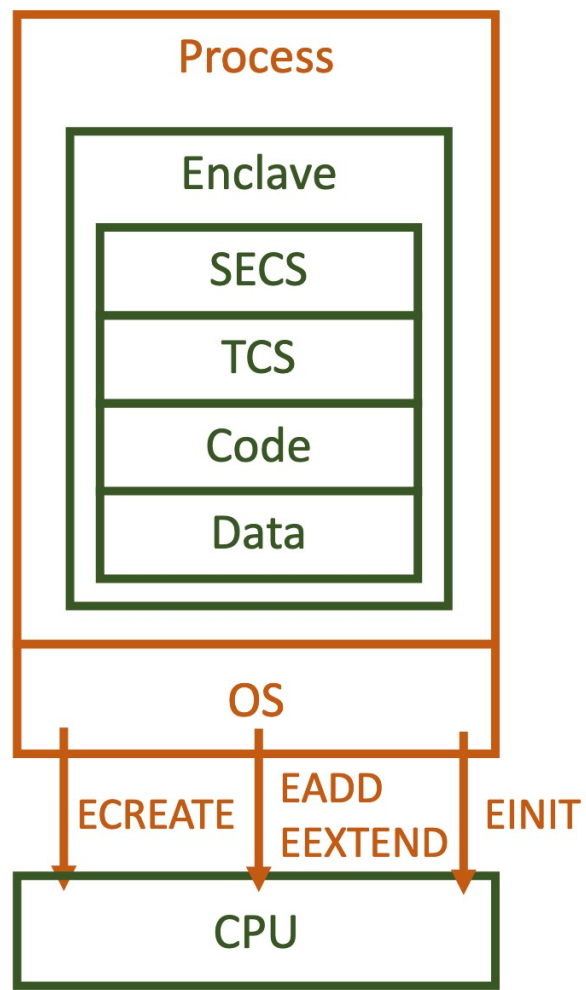
▣ Thread Model



▣ Elements

- ◆ Secure Boot
- ◆ On-chip program isolation
- ◆ Cryptographically protected external memory
- ◆ Execution integrity
- ◆ Attestation

Example: Intel SGX Enclave creation/enter Exit



Example: AMD SEV Kvm case

- ▣ `GRUB_CMDLINE_LINUX_DEFAULT="mem_encrypt=on
kvm_amd.sev=1"`
- ▣ `ubuntu@nsXXX:~# dmesg | grep SME [1.247928] AMD
Secure Memory Encryption (SME) active`

Limitations of TCBs and TEEs

▣ Vulnerabilities in TCBs

- ◆ Current susceptible to TCB-resident attacks: SMM-based and ME-based rootkits
- ◆ Unable to get-rid of it (from the administrator perspective)
- ◆ Once TCB is compromised, TEE is no longer secure: e.g., **foreshadow**

▣ Opaque TCB execution

- ◆ Often there is no means for auditing the code executed by TCB
- ◆ Proprietary code (usually trade secret) with infrequent updates and signed
- ◆ Code running TEE should be fingerprinted continuously (i.e., attestation via hashing or performance signature)
- ◆ Its not the case: in closed hardware it's a closed box != open hw (riscv) **Keystone-enclaves**

▣ TEE-Based Attacks

- ◆ Use the TEE as an attack vector: e.g., **SGX-Bomb, plundervolt**

▣ TEE Code Bloat

- ◆ Not a good idea to increase the size of the code inside the TEE (e.g., run containers inside SGX or a Hypervisor running inside SMM are proposed for flexibility).