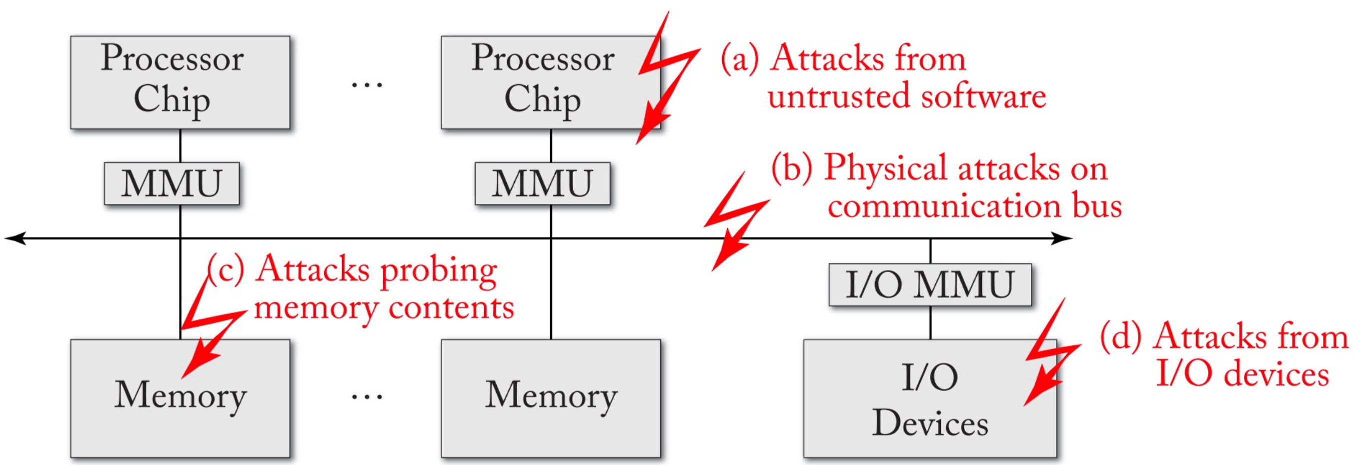
# Memory Protection

Chapter 6

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

# Threats Against Main Memory

- Wiring and memory device itself are untrusted assumed built upon DRAM devices
- Sources of Attacks
  - Untrusted software (bypassing isolation barriers)
  - Malicious devices (trying to access via DMA to protected regions)
  - Physical attacks (on memory "bus" via probing)
  - Physical attacks (on memory itself, Cooldboot, Rowhammer)
- Memory can be easily removed and analyzed off-line
- Competing Non-volatile Memories (NVM) in the horizon (such as Intel Optane) can make this problem harder



# Attacks to Memory

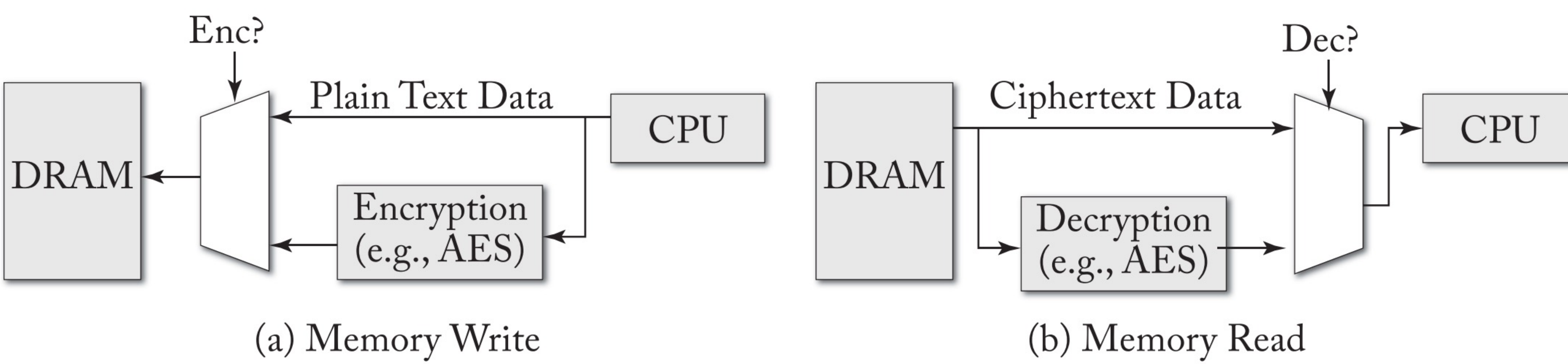
- Passive Attacks
  - **Eavesdropping attacks**: observe information or accessing patterns without altering it to gain knowledge
    - Pattern: V.gr. Observing AES S-Box access pattern can leak information about the encryption key
    - Prevented: **encryption** and **obfuscation**
- Active Attacks
  - **Spoofing attacks**: inject memory data (or operations) without being detected
    - Inject changes in memory mapping, processor instructions, stole data, …
    - Prevented via **hashing**
  - **Splicing Attacks**: combine multiple read/write operations in a new (legitimate) read or write
    - Splice parts of different messages (e.g., payload from one and header from other)
    - Prevented via **keyed hashing** (MAC Message Authenticated Code)
  - **Replay Attacks**: Send messages again
    - Reuse old "known" messages (v.gr. replace an encryption key)
    - Prevented via **nonced hashing**

# Main Memory Protection Mechanisms

- Three main techniques and objectives
  - **Confidentiality**: with encryption
  - **Integrity**: with hashing (typically hash-trees)
  - **Access Pattern protection** with access pattern obfuscation
- Memory protection is focused on off-chip related issues
  - Encryption and/or hashing is only done at **off-chip processor interfaces**
  - Inside the processor chip regular isolation techniques (e.g., page tables for equally privileged software or tagging for higher privilege software)
  - Might increase average access time
  - Access pattern still discoverable
- MMU and IOMMU are also useful

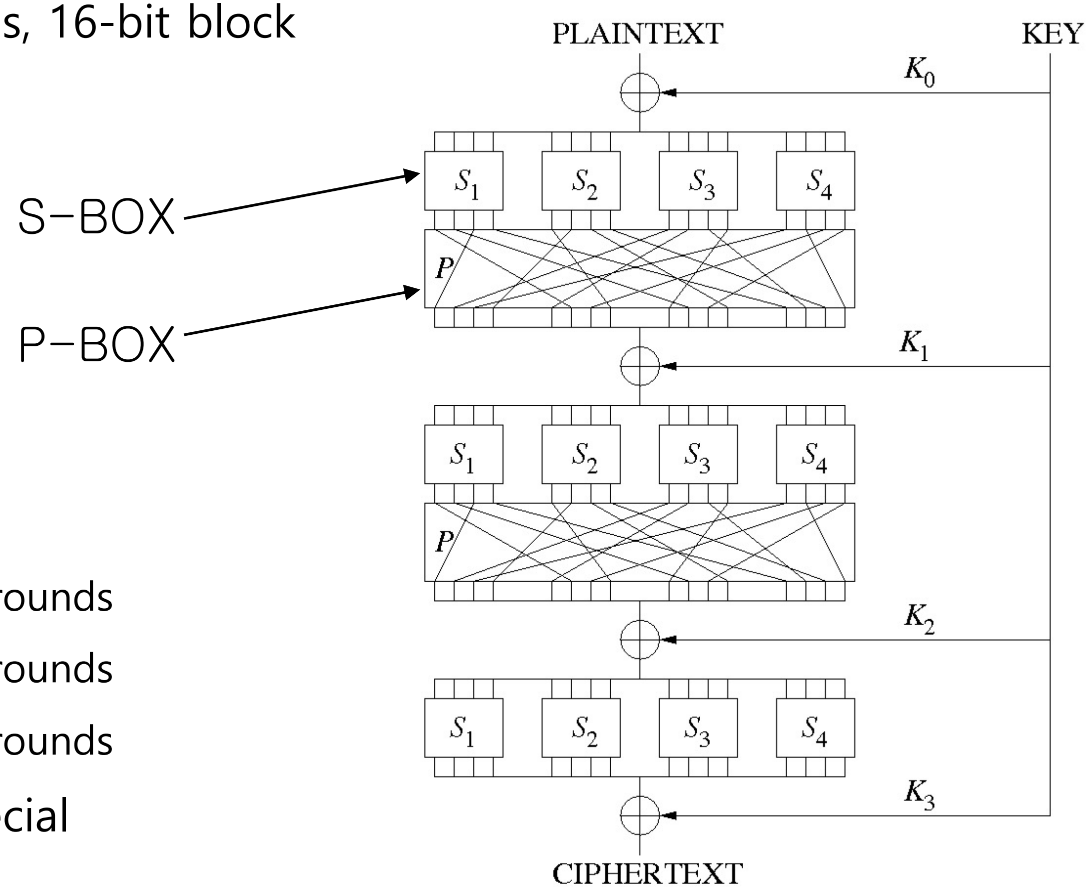
# Confidentiality Protection With Encryption

- Use a **re-generated** key each time system **reboot**
- Store the key in the MMUs or special management module (usually Memory Controller)
- Application, OS or system level. Can be **selectively** enabled in some ranges of the addressing space of the process or physical addresses of the memory (use ASIDs in tags to isolate)
- Use asymmetric cryptography to support multi-socket systems (the key never should leave the processor chip in plaintext!!) or use independent key per memory controller/ASID
- Memory access will be slower for encrypted data (use **suitable algorithms** for the task: AES CTR Mode)
- To support DMA IOMMU also should have a key (I/O and PCIe is unencrypted, therefore vulnerable against probing)
- Can be vulnerable if management engine is untrusted (will have access to encryption keys)
- AMD Secure Encrypted Virtualization implements it



# Aside: AES Algorithm (10.000 feet view)

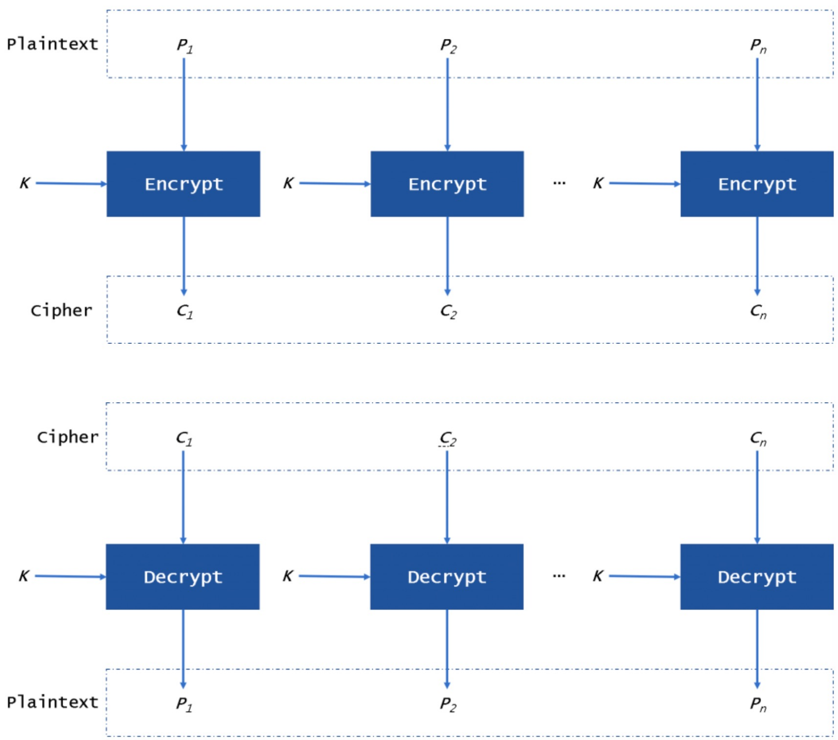
- Base on substitution-permutation network
  - Example 3-rounds, 16-bit block

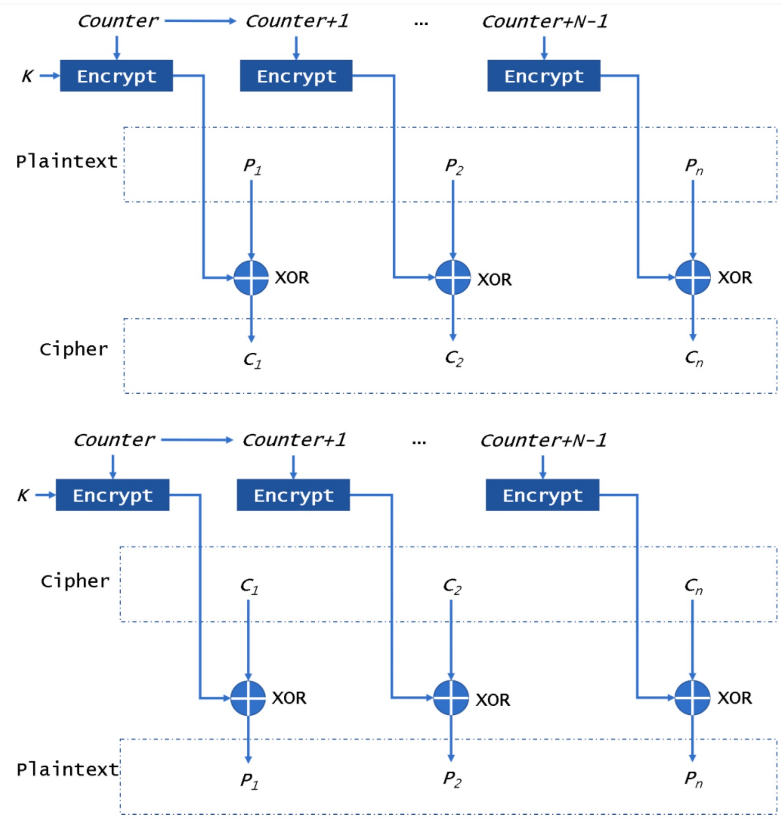



- AES
  - 128-bit blocks
    - 128bit key: 10rounds
    - 256bit key: 14rounds
    - 192bit key: 12rounds
  - Last round is special
- AES-NI
  - Instructions for one round of ENC/DEC

# Aside: Counter Mode AES (fast encryption/decryption)

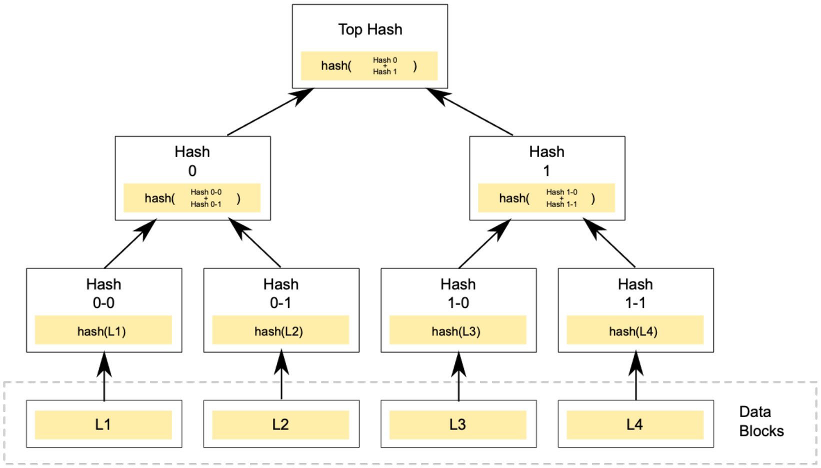
- Electronic Code Book vs Counter Mode (AES has 5 modes)



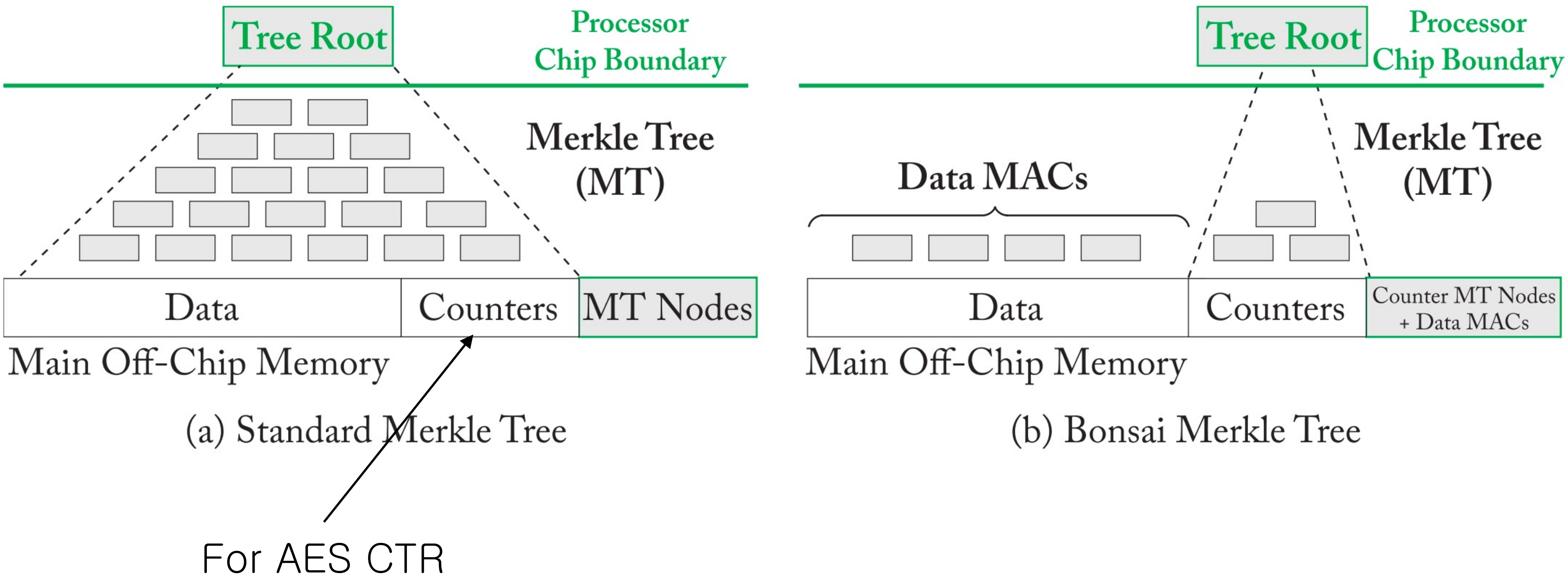
# Integrity Protection With Hashing

- Cryptographic hashes
  - Produces a unique output for an input (negligible probability of collision of two inputs if the hash is long enough)
  - We can't figure out the input from the output (non inversible functions)
- The result of hashing the data can be understood as a fingerprint of the data
  - The easiest way to check if the data has been altered is check hash result
- Integrity protections focused on **external attackers**
- Hash the whole memory is **impractical**: slow and every change requires to recompute the hash: **Use hash trees**
- Log2(N) hash computations (N: depth of the tree)
  - **Write**: update all the hashes from the leaves updated to the root
  - **Read**: check if involved hashes to the data are ok
- But attacker can modify the whole tree (??)
  - Add encryption to the hash



# Merkle Trees

- Root of tree on secure on-chip register



# Hash Tree Protection

- Protect intermediate nodes in the tree: attacker can compute and insert its own hashes,
  - Do not use plain hashes: use secure processor-based keys to encrypt the hash (the tree is in memory!)
  - Use cryptographic hashes: **keyed hashes** (**MAC** or Message Authenticate Code) and always on secure on-chip locations: can’t be tampered (but can be cached)
- MAC can protect against splicing and spoofing but **not replaying**
  - Add a monotonic counter for **replay attacks** protection
- **Secure page swapping** might require to reserve leaf nodes in the hash tree for non-present pages (i.e., integrity protection trees should cover the disk)
- Variety of **performance improvement** to minimize hashing performance impact
  - Counter mode encryption (out of the critical path)
  - Bonsais Merkle independent encryption seed from the memory address (do not combine key with address in the original tree)
- Intel SGX uses a tree of MACs for integrity and replay protection

# Access Pattern Protection

- Just observing the access to address activity can be used as conduit for revealing secrets
- ORAM (Oblivious RAM) (1984) keeps the semantics of the program but hides the access pattern by **shuffling memory locations** (inserting noise at compile time)
  - On each access memory insert random access. The objective is to make indistinguishable to the attacker the useful accesses from the noise
  - Only in sensible data (software + hardware, e.g. S-Box access in AES encryption)
- All OS does something called Address Space Layout Randomization (ASLR) but oriented to prevent exploits. Its at page level