看懂可信执行环境硬件设计

- 浅析TEE的内存加密和完整性保护机制

中国科学院信息工程研究所
INSTITUTE OF INFORMATION ENGINEERING,CAS

王文浩

自我介绍

王文浩

信息工程研究所，信息安全国家重点实验室

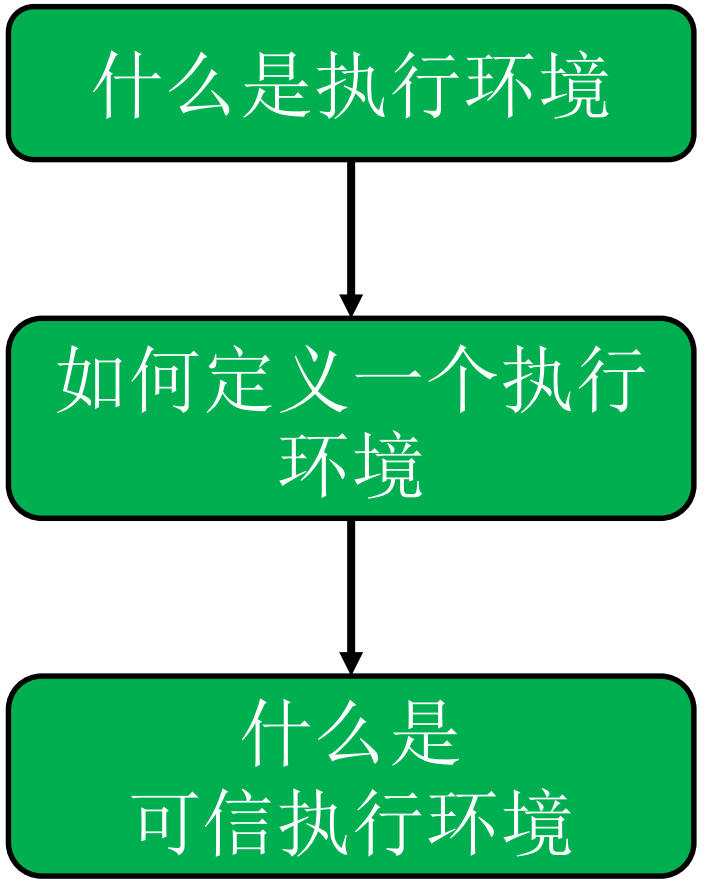
副研究员，硕士生导师

主要从事系统安全、侧信道攻防、TEE等方向研究，曾发现SGX多个侧信道泄露途径和泄露方式，提出SGX侧信道防御的方案，设计了支持异构硬件的TEE架构。部分成果发表在CCS（2017）、S&P（2018, 2020, 2021）、EUROCRYPT（2018）、IEEE TDSC、IEEE TIFS、ACSAC（2018）、CHES（2020）、DSN（2021）等知名国际会议和期刊。担任CCS 2019程序委员会委员，获得ACM中国新星奖提名奖和ACM中国SIGSAC分会新星奖。

主页： https://heartever.github.io

邮箱： wangwenhao@iie.ac.cn

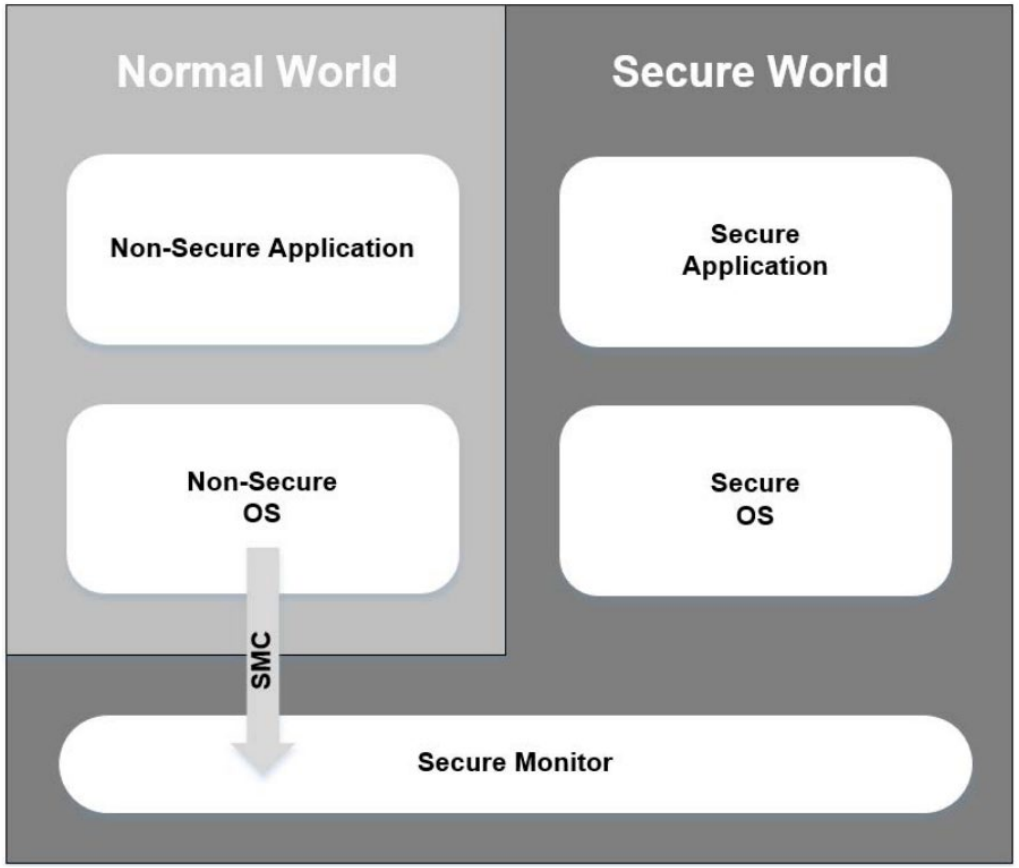
什么是可信执行环境



TEE是怎么设计出来的?

- 提供什么样的（抽象）执行环境
  - 进程、OS、安全世界
  - 运行模式、指令集等
  - 如何处理中断、异常
    - CPU调度、内存管理（缺页）、非法指令
  - 如何与外界交互
    - 接口/IO等

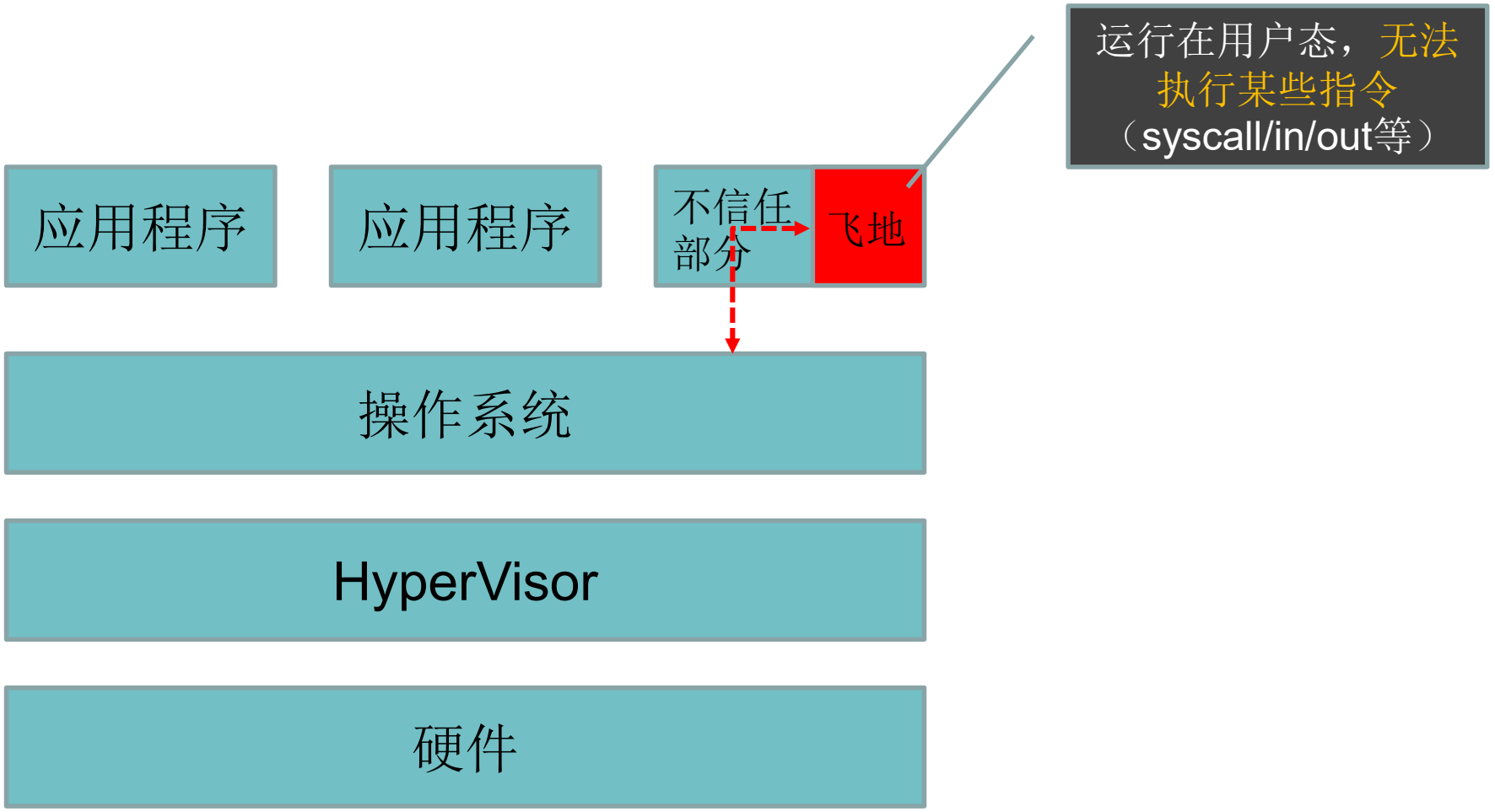
TEE的不同抽象



基于系统级的硬件隔离机制实现2个执行环境

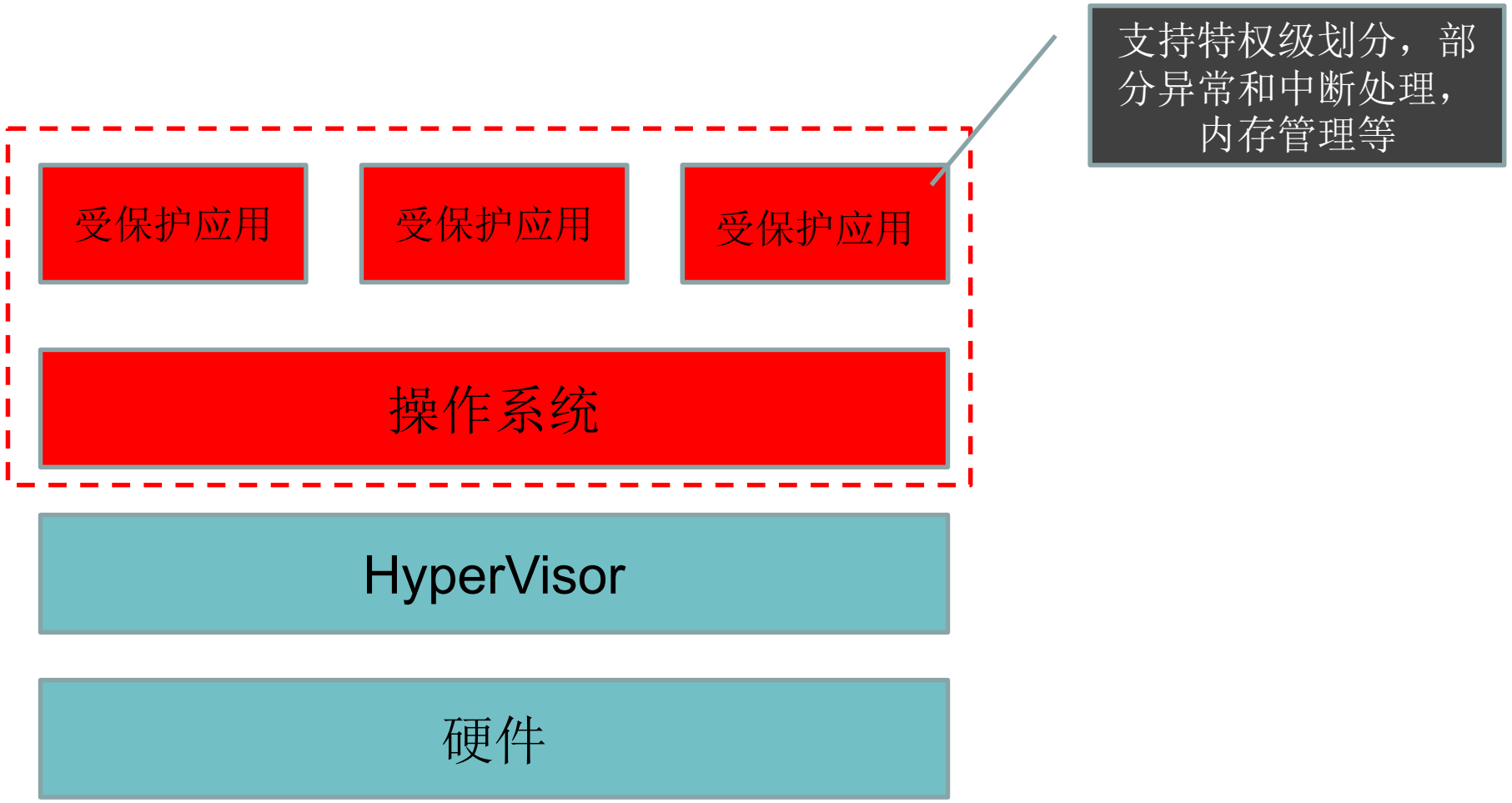
系统级：ARM TrustZone

TEE的不同抽象



进程级：Intel SGX

TEE的不同抽象



虚拟机级：AMD SEV等

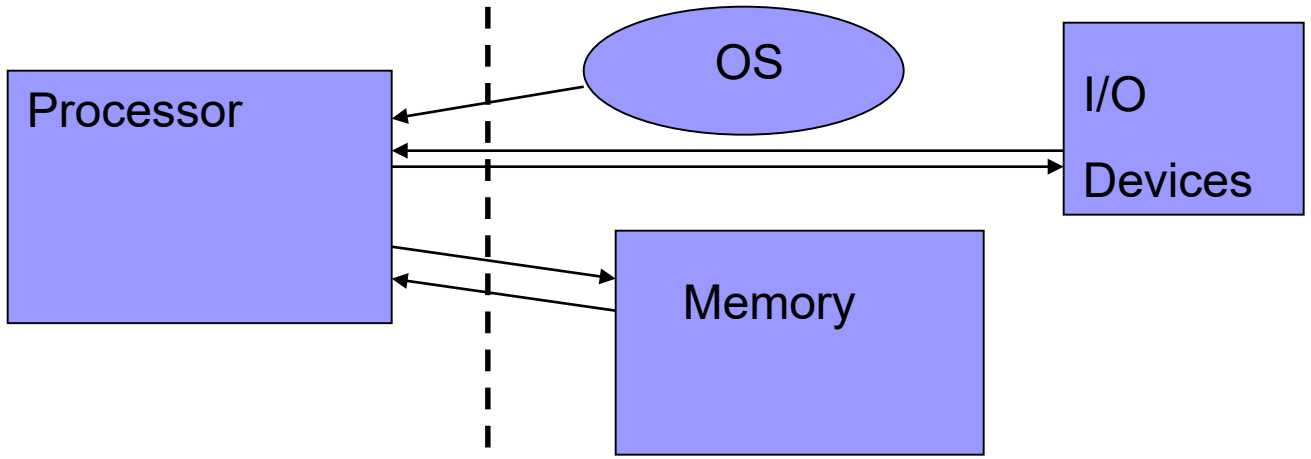
TEE是怎么设计出来的?

- 提供什么样的（抽象）执行环境
  - 进程、OS、安全世界
  - 运行模式、指令集等
  - 如何处理中断、异常
    - CPU调度、内存管理（缺页）、非法指令
  - 如何与外界交互
    - 接口/IO等

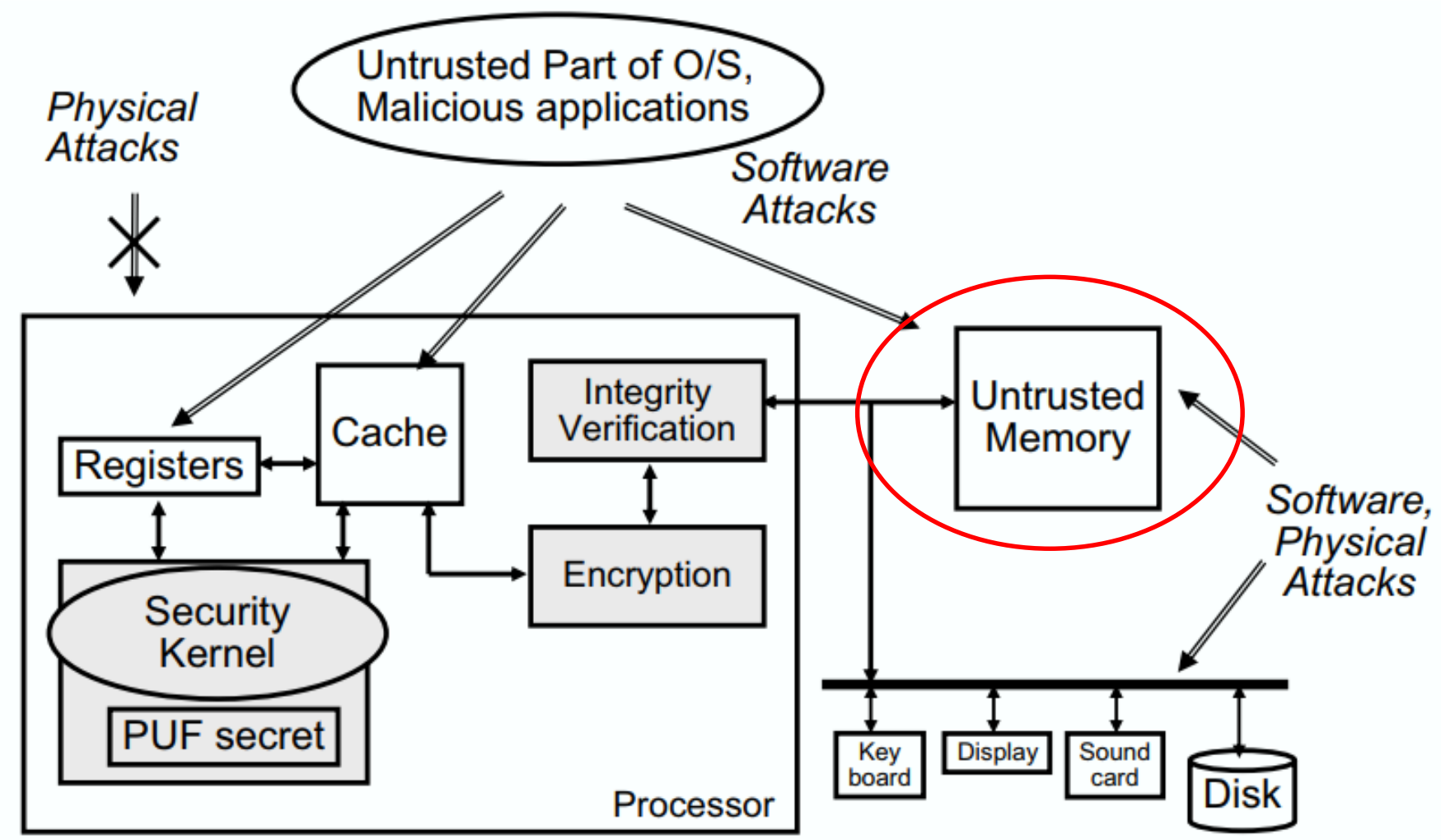
- **考虑的威胁模型是什么？**
  - 物理攻击、软件攻击、侧信道攻击
  - 如何实现隔离机制？

威胁模型

- 假定处理器和片上存储是安全的
- 操作系统和所有的外围设备，包括片外内存是不信任的



攻击面


Untrusted Part of O/S,
Malicious applications
Physical
Attacks
Software
Attacks
Registers
Cache
Integrity
Verification
Untrusted
Memory
Encryption
Security
Kernel
PUF secret
Processor
Software,
Physical
Attacks
Key
board
Display
Sound
card
Disk

以保护片外存储为例

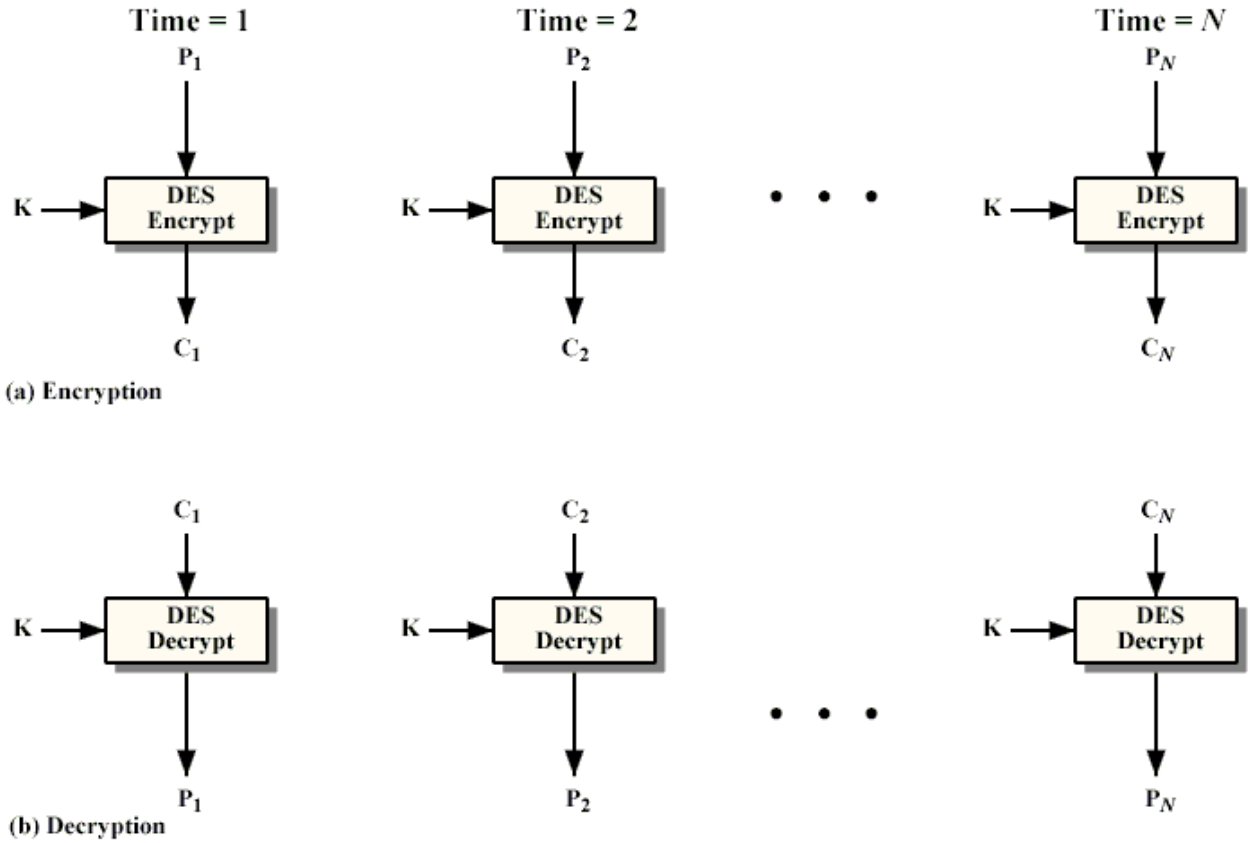
- 攻击者通过操纵片外的未保护的内存
  - 直接观察
    - **解决方案：内存加密**
  - 篡改数据
    - 任意修改（spoofing），拼接（splicing），重放（replay）
    - **解决方案：完整性保护和防重放**
  - 观察地址总线的访问模式
    - 侧信道
    - **解决方案：ORAM**

机密性

- 内存加密模式
  - AES+ECB/CBC/CTR/XTS/GCM
  - 可调分组密码：QARMA

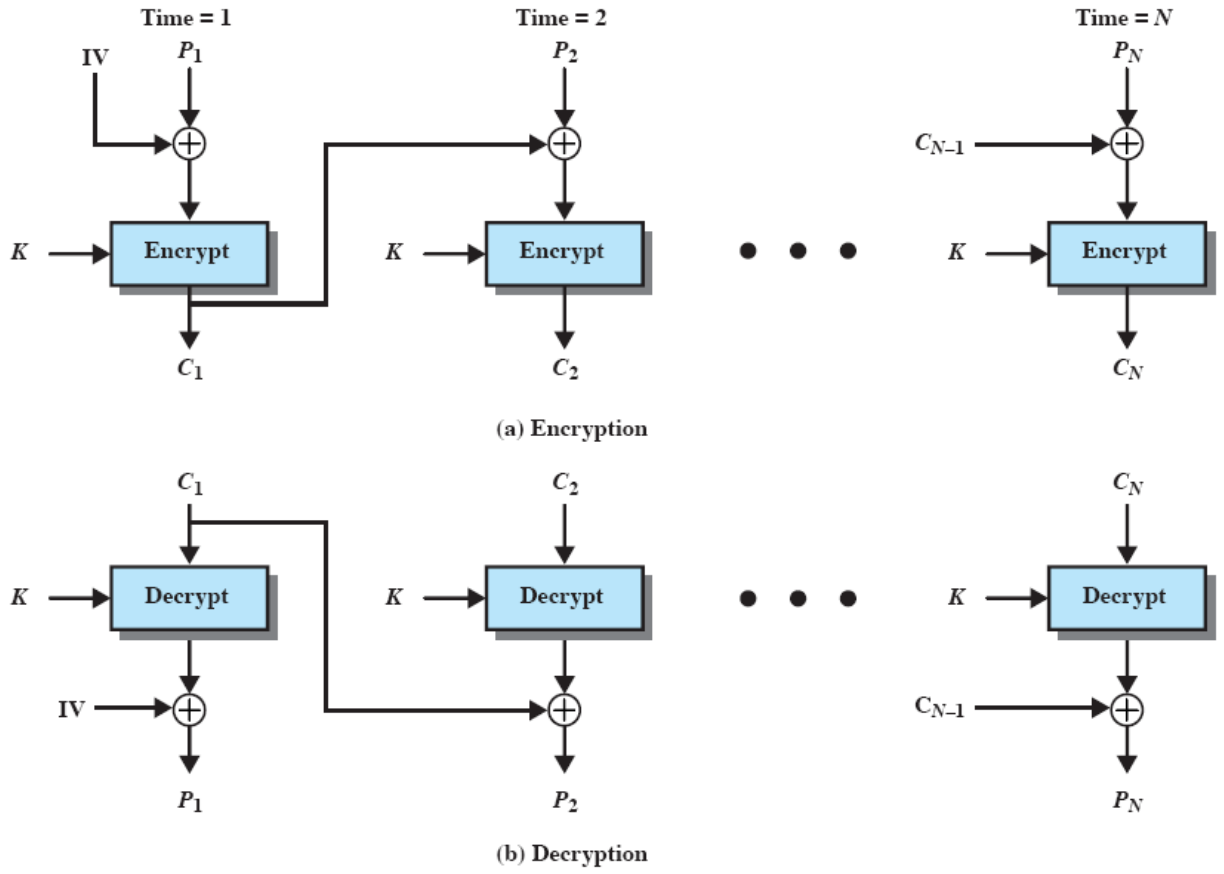
ECB

- 用相同的密钥分别对明文组加密
- 各个分组独立加密和解密
- 频率分析攻击
- 密文替换攻击



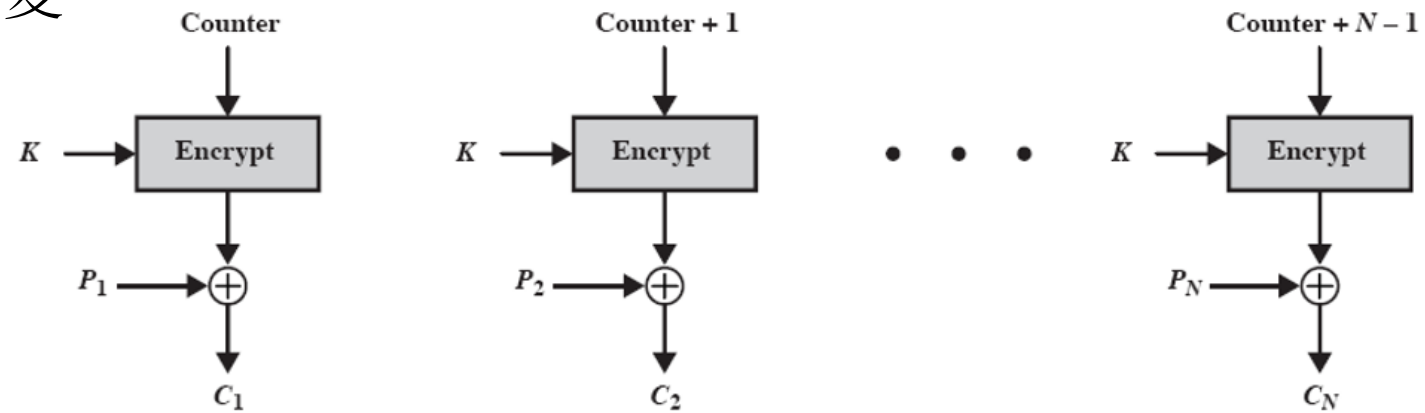
CBC

- 当前明文分组先和前一个密文异或，再加密
- 初始向量IV，不必保密，但需要保存并占据空间
- 无法并行

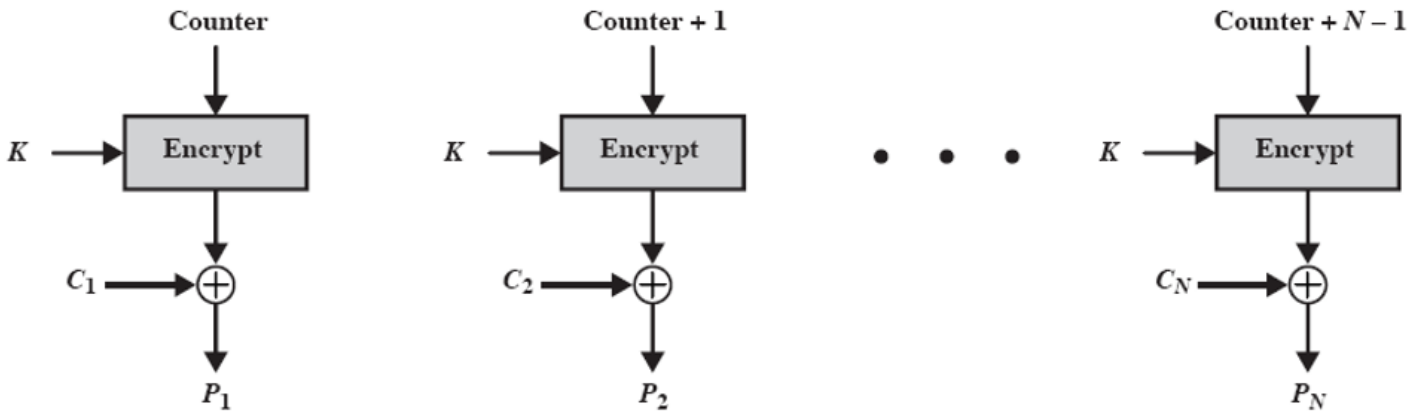


CTR

- 可以并行,counter占据存储空间
- counter不能重复



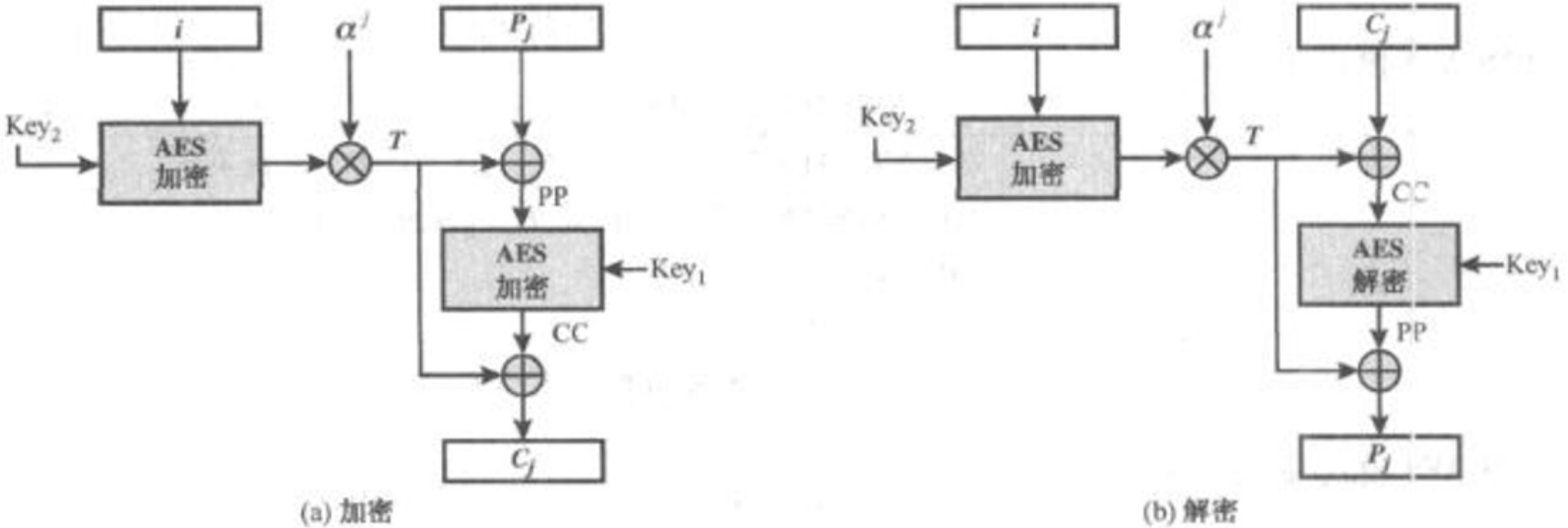
(a) Encryption



(b) Decryption

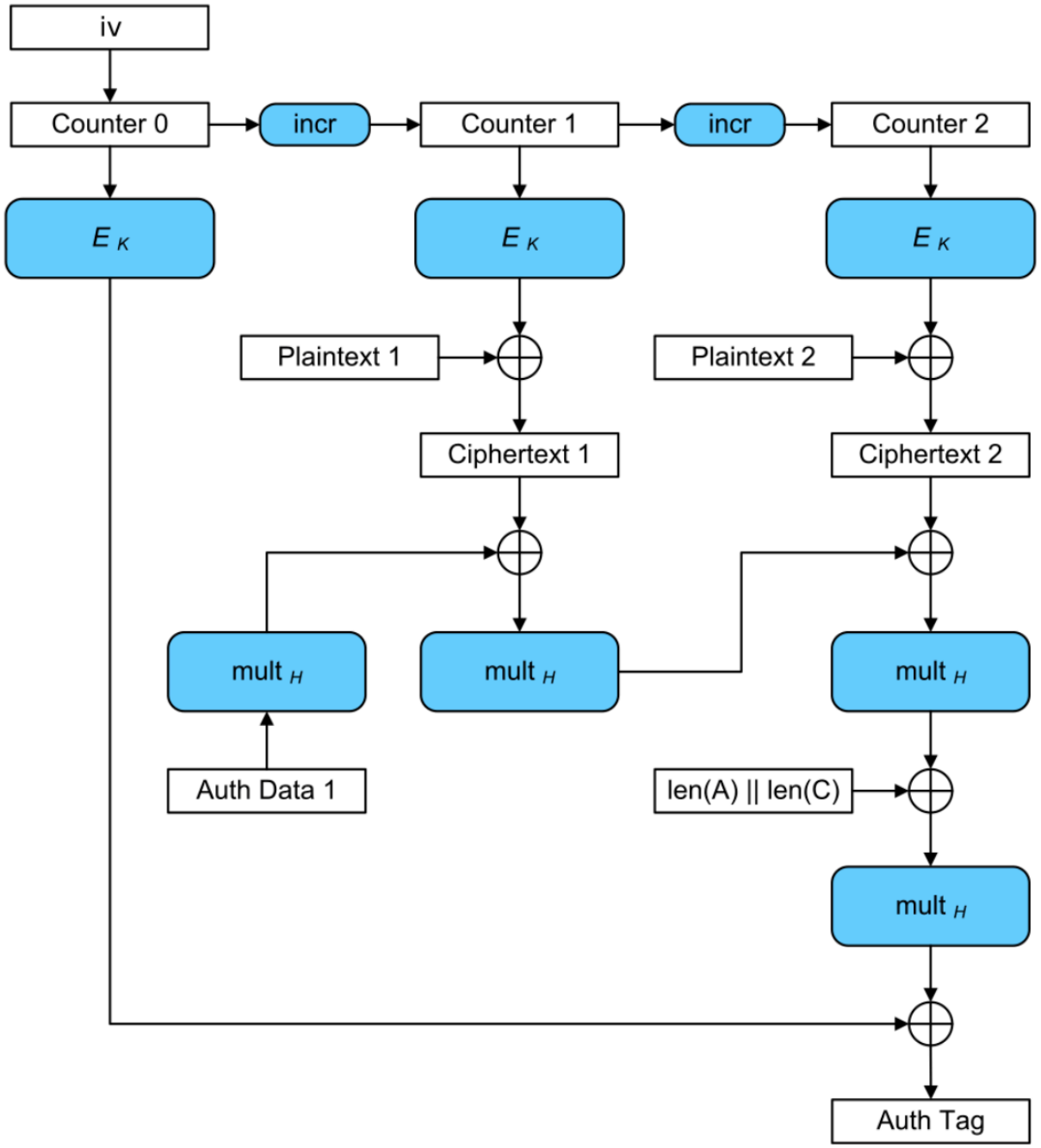
XTS

- 明文和密钥之外，算法包括一个tweak作为输入
- 相比CBC、CTR等模式，不需要额外的存储IV和counter
- 整体与ECB类似，例如可以并行
  - 但规避了ECB的缺点：频率分析攻击，密文替换攻击



完整性

- MAC
  - SHA1/SHA3……
- 认证加密模式
  - CTR+GMAC
  - Counter需要保存、不能重复
- 认证加密算法
  - CAESAR竞赛认证密码算法征集

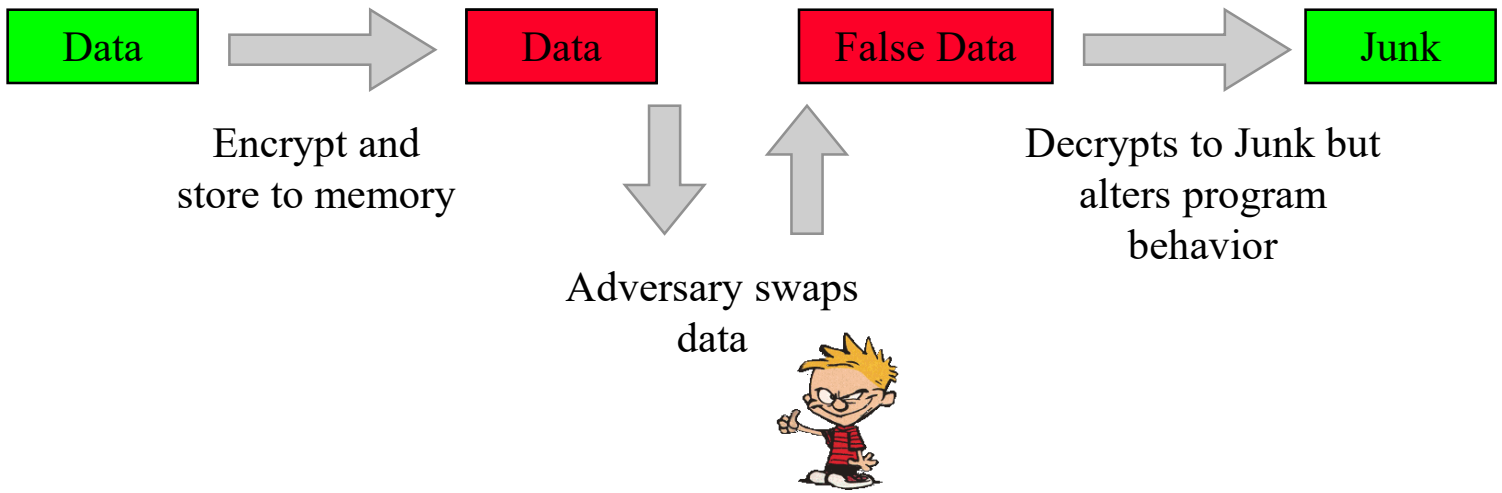


Xom – ASPLOS 2000

- 机密性
- CTR模式
  - cipher = plain ⊕ encrypted$_{key}$(address || seq)
  - plain = cipher ⊕ encrypted$_{key}$(address || seq)
    - *key* = XOM ID
    - *address* = virtual address of data/instruction
    - *seq* = mutating sequence number
- encryptedkey(address + seq)的计算可以与内存访问并行
- 最终的计算只需要1个cycle进行异或

仅加密是不够的

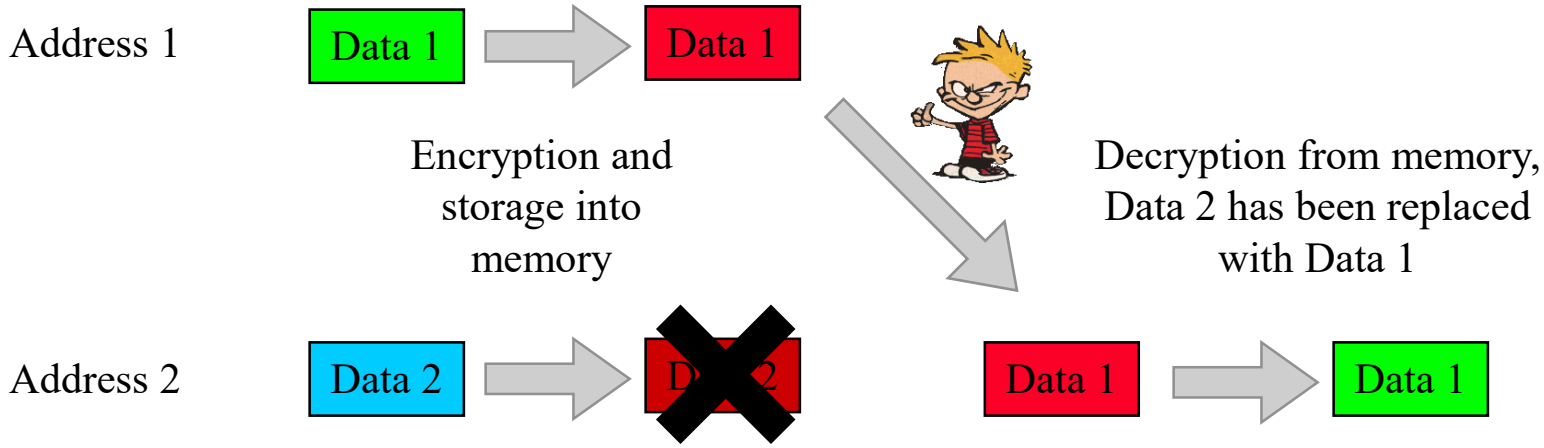
- 解决方案：在加密结果之外附加保护完整性的哈希（MAC）
- Adversary has to reverse encryption to fake the hash



为什么需要address作为counter的一部分

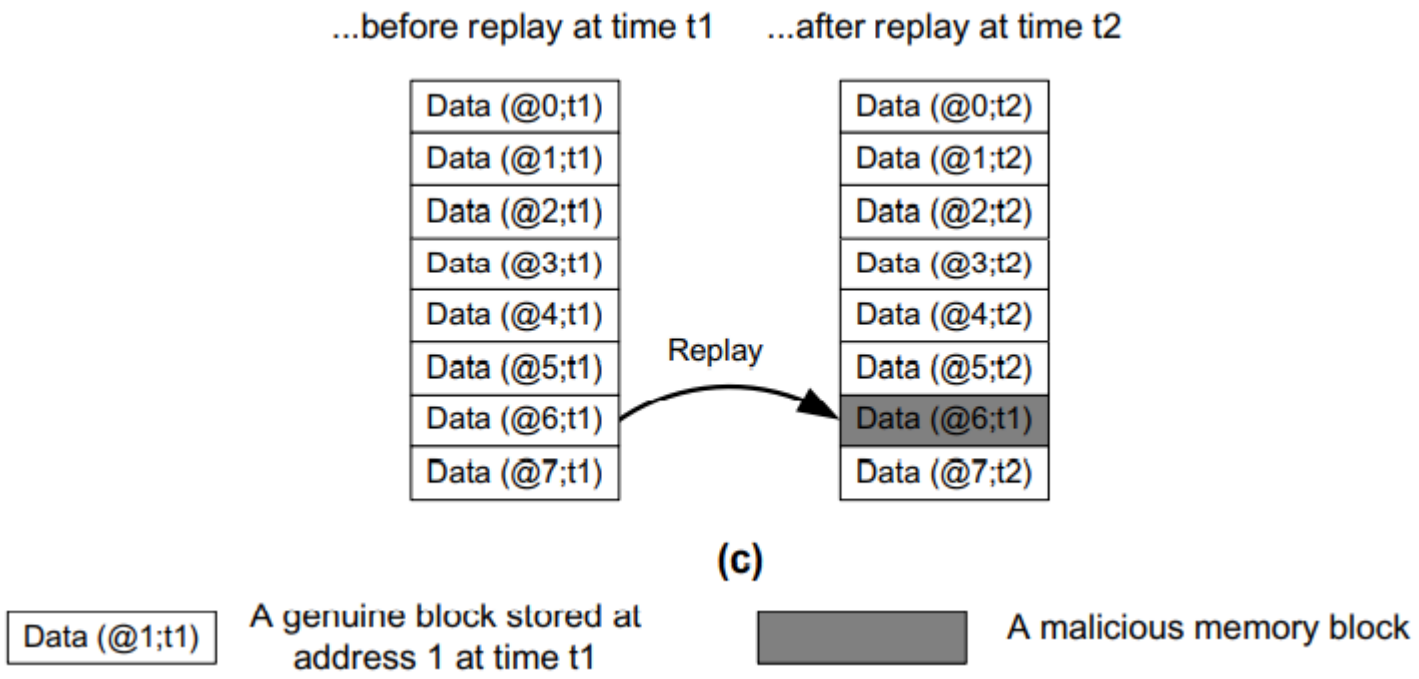
- 从一个内存地址复制合法的密文，替换到另外一个内存地址

$$cipher = plain \oplus encrypted_{key}(address \| seq)$$
$$plain = cipher \oplus encrypted_{key}(address \| seq)$$



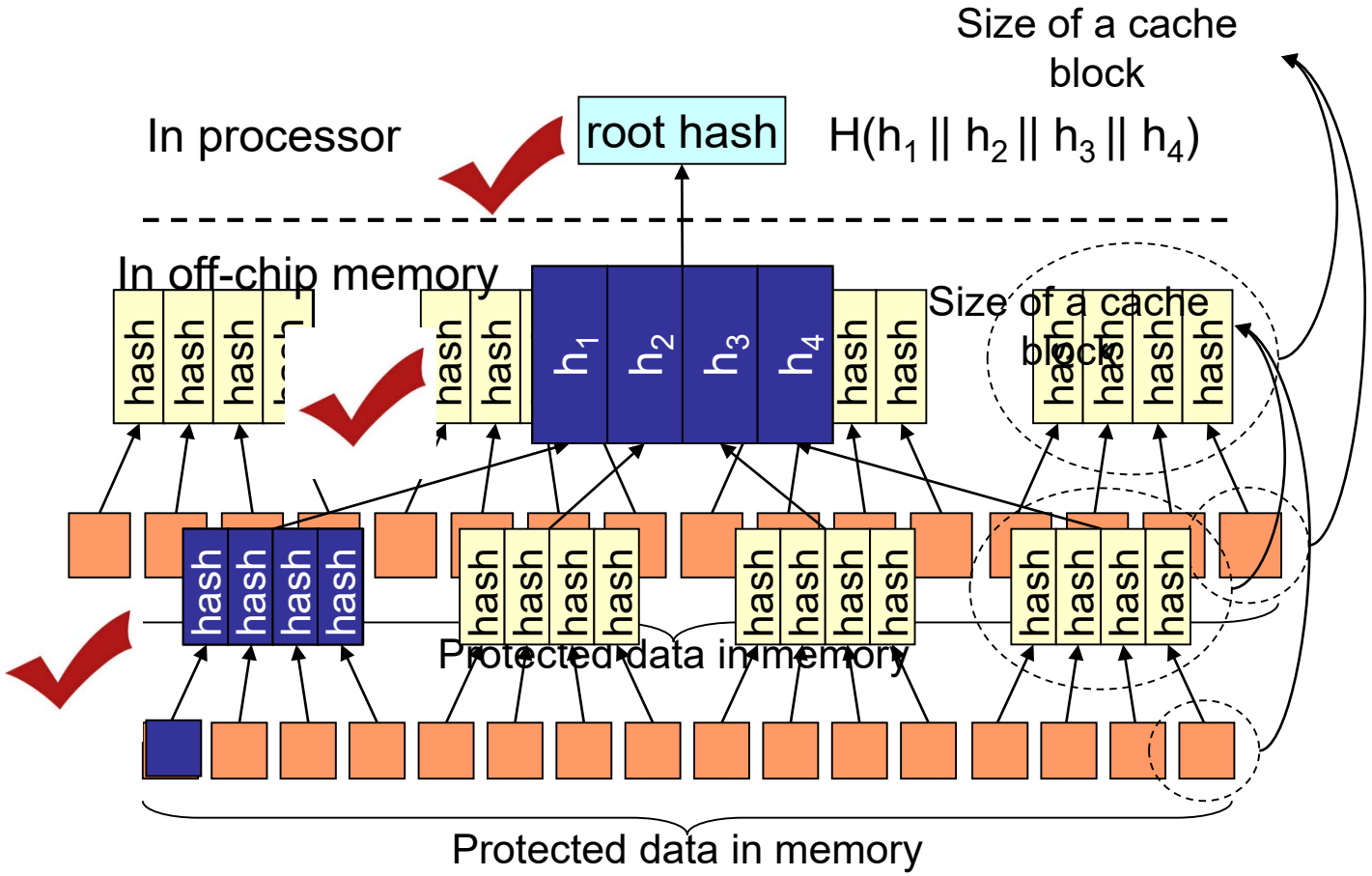
Xom无法提供新鲜性保证

- 使用同一地址的旧的数据替换该地址的新的数据



新鲜性-防重放

- Merkel Tree



Caches and Merkle Trees for Efficient Memory Integrity Verification. HPCA 2003

优化

- 若每次都计算到根节点
  - 4GB内存，128-bit hash，性能损失可以达到10倍

- 专门的cache结构，缓存了最近使用的内部hash块
  - 不需要计算到根节点，只需要计算到查找到的中间节点
  - 若中间节点被换出，则将其父节点放入cache，并更新父节点
  - 平均22%，最坏52%

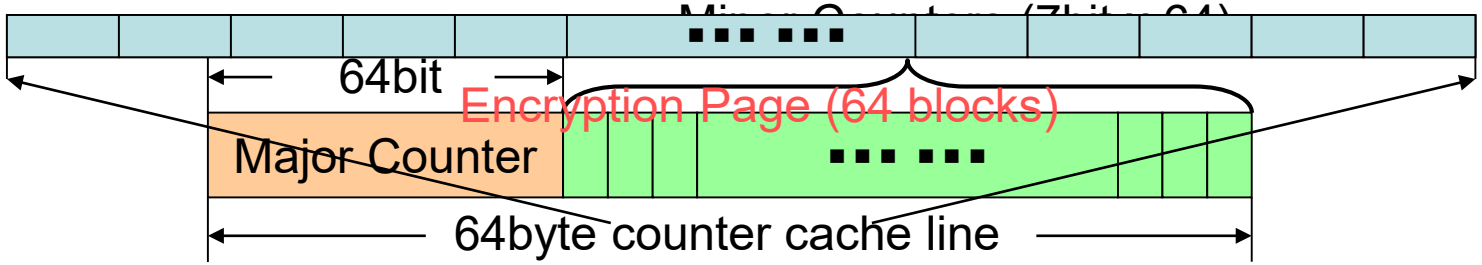
Counter的压缩模式

- CTR的问题
  - 每次cache write back需要增加一次counter
    - 每个chuck一个counter+addr
  - 当counter溢出时，需要换一个AES key
    - 但整个内存加密只有一个key，因此需要重新加密整个物理内存 – freeze the system

Improving Cost, Performance, and Security of Memory Encryption and Authentication. ISCA 2006

Counter的压缩模式

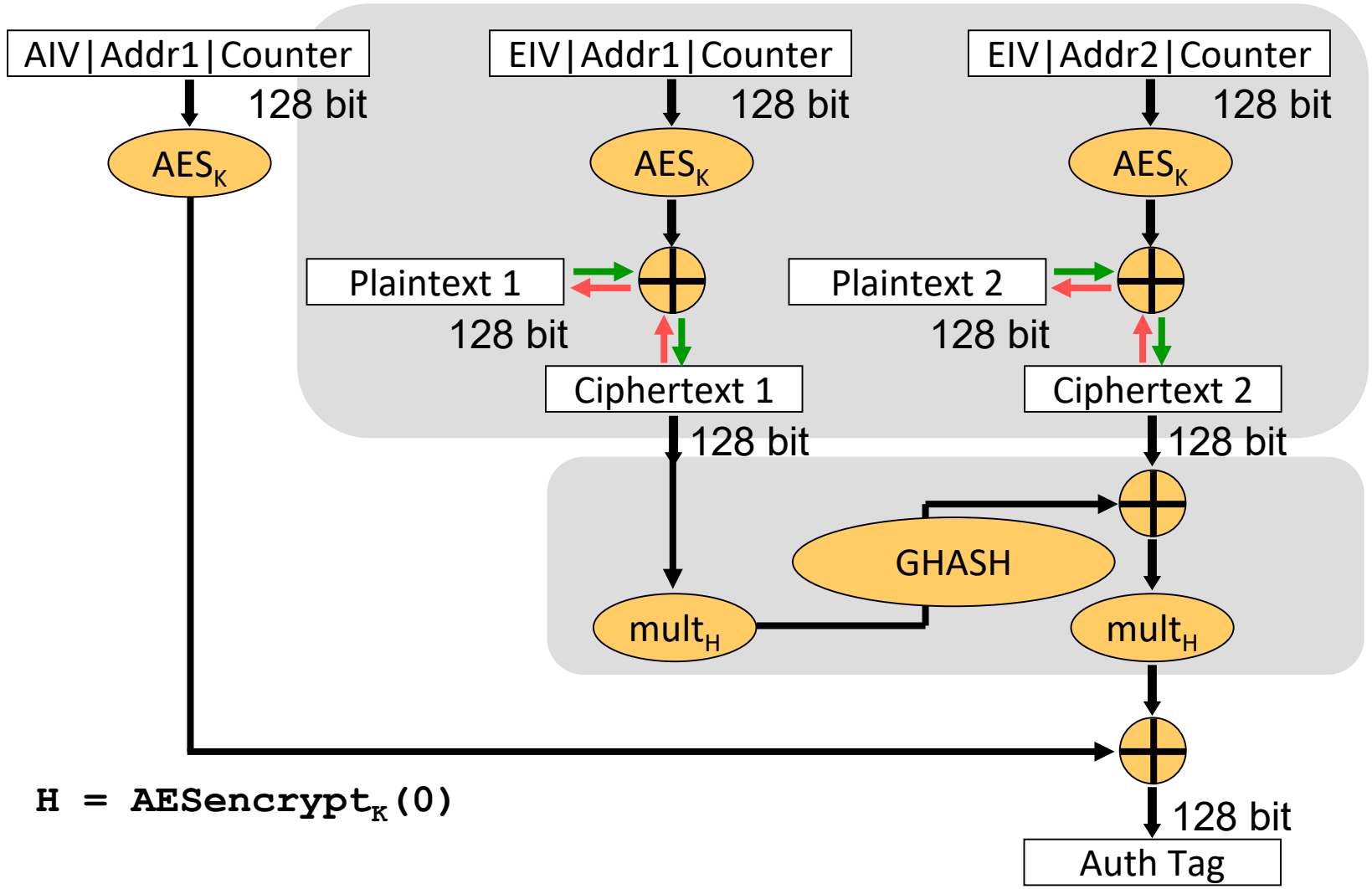
- Counter = Major Counter || Minor Counter
  - Major Counter
    - 每个加密的页面内所有的数据块共享一个major counter
    - 在系统的生命周期内都不会溢出（64 bit）
  - Minor Counter
    - 页面内每个数据块具有的独立的counter
    - 长度较短，容易溢出，但溢出时只需要更换一个major counter，并重新加密该页面，而不是全部的物理内存
- Stored together in a counter cache line
  - 存储开销为1/64=1.6%



Improving Cost, Performance, and Security of Memory Encryption and Authentication. ISCA 2006

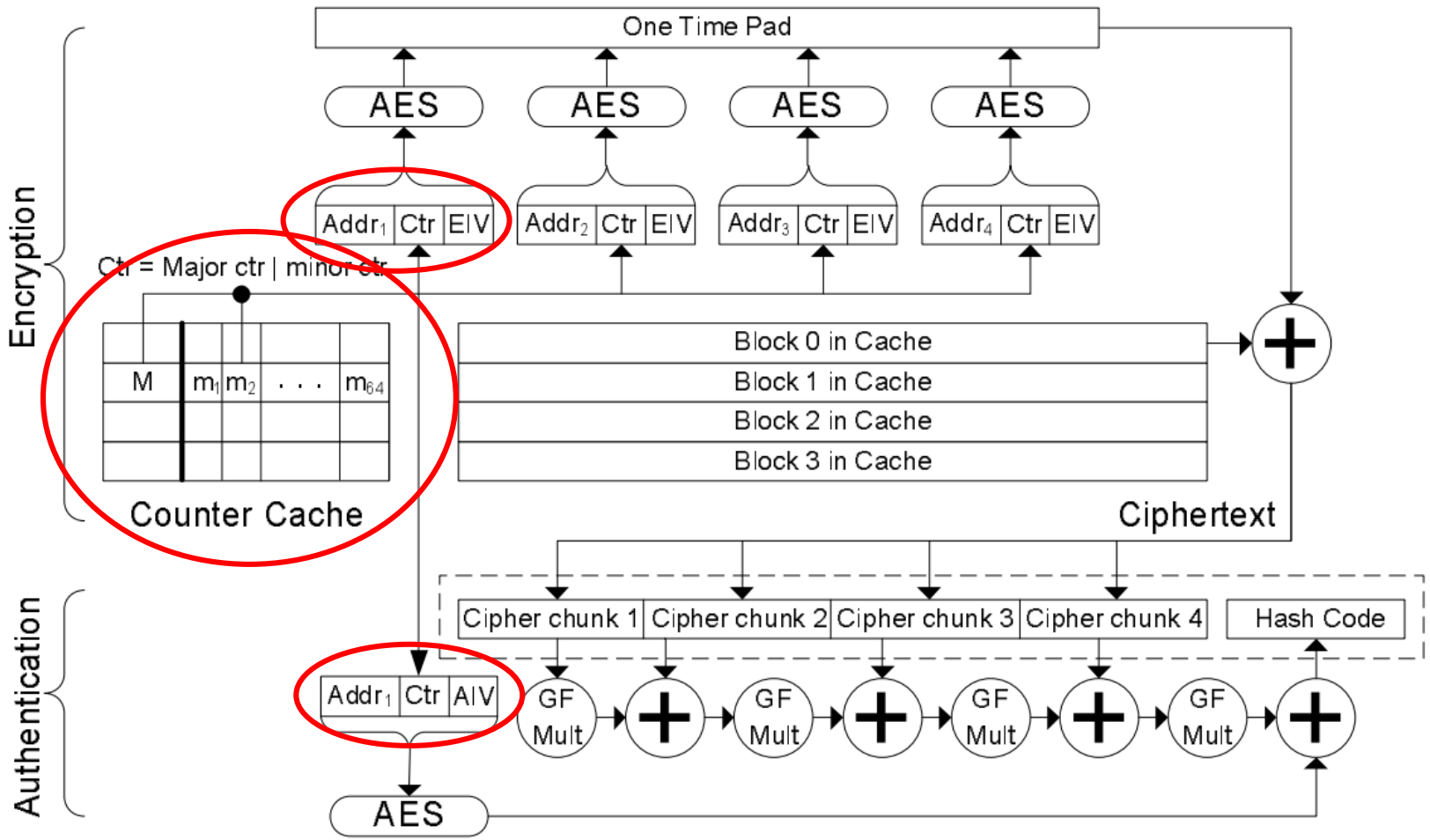
优化完整性保护

- 使用GMAC



Improving Cost, Performance, and Security of Memory Encryption and Authentication. ISCA 2006

整体方案



Improving Cost, Performance, and Security of Memory Encryption and Authentication. ISCA 2006

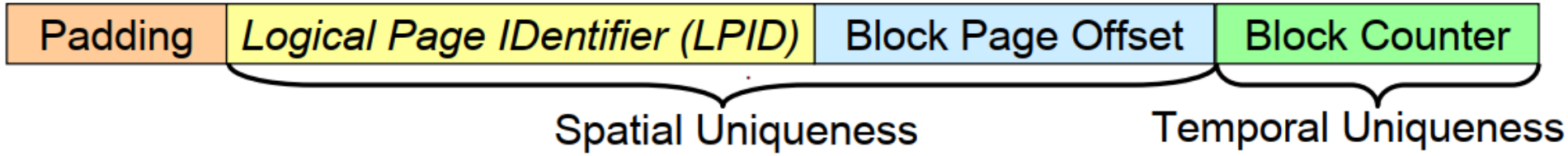
继续优化：motivation

- 内存加密
  - CTR模式可以隐藏内存访问关键路径的延迟，但是该模式下要求counter是唯一的
    - 之前的工作，采用了地址（spatial uniqueness，va或pa）作为counter的一部分
    - 每次write操作，counter+1，用于保证temporal uniqueness
- 问题
  - 使用物理地址，则每次page swapping都需要重新加密和解密
    - 且需要增加额外的完整性保护机制
  - 使用虚拟地址，但不同的进程可以使用同样的虚拟地址
    - 若把每个进程的ID也作为counter的一部分
      - 无法支持shared memory based IPC和shared library
      - process fork的copy on write的优化不可用，原因是parent/child的counter是不一样的
      - 进程ID是由不信任的OS指定的

Using address Independent Seed Encryption and Bonsai Merkle Trees to Make Secure Processors OS- and Performance-Friendly. MICRO 2007

主要优化

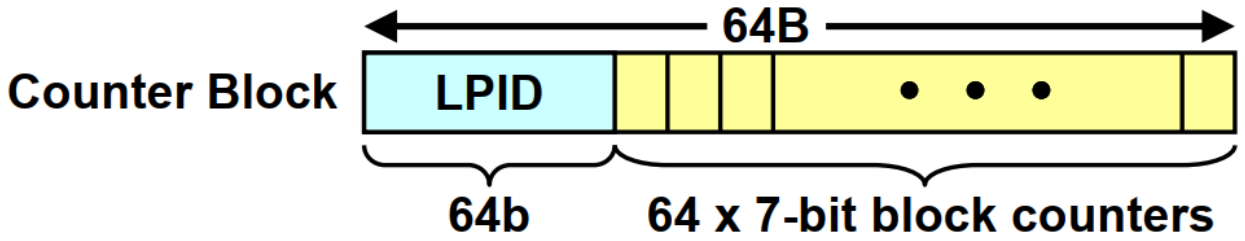
- Address Independent Seed Encryption （AISE）
  - 为每个页分配一个逻辑识别码（logical identifiers）；与虚拟地址或物理地址无关，在整个系统的生命周期确保对所有的内存页都是不同的
    - 可以避免page swapping等的性能问题
    - 可以支持shared memory/shared library等机制



- LPID
  - 在页分配时分配的唯一值（unique value）
  - 在生命周期（系统启动到关闭）内与页绑定，包括物理页和交换空间页
  - Global Page Counter：64比特
    - 无法篡改，几乎不会溢出

主要优化

- LPID的存储：与压缩counter结合



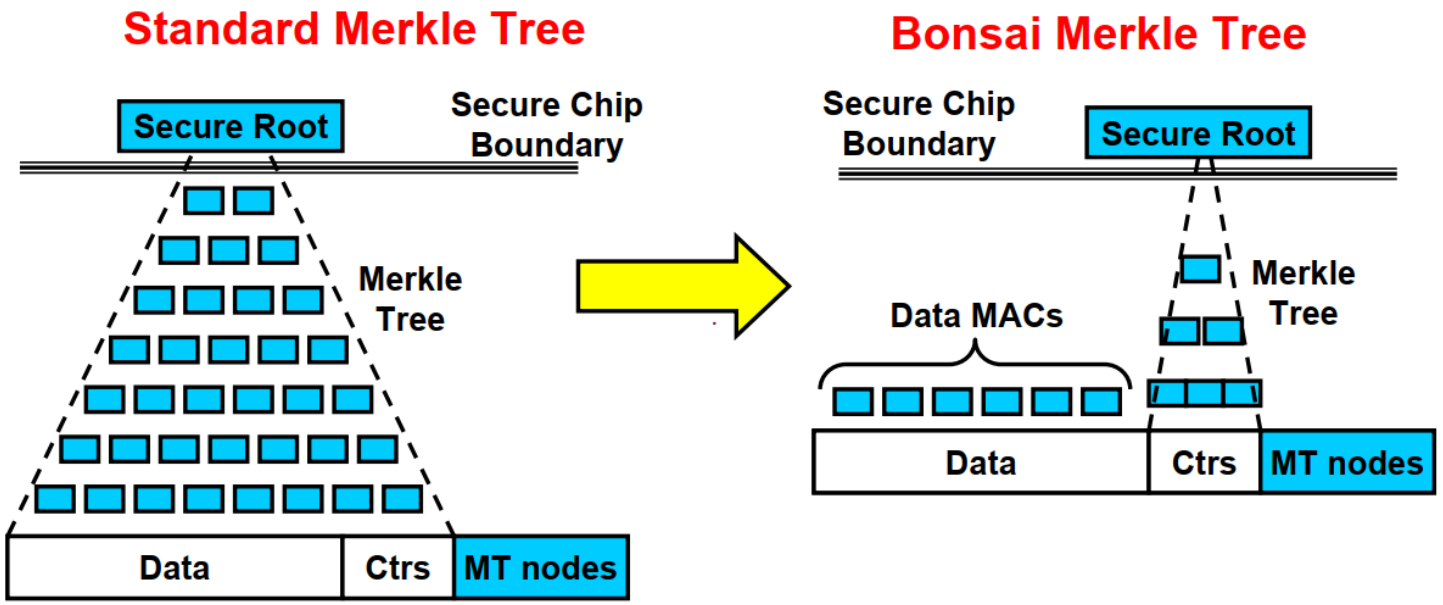
- 以1个4KB的页为例，每个block为64字节，每个block的LPID长度为64比特，每个block分配一个7比特的counter
  - 若block counter溢出，则为该页分配一个新的LPID，并重新加密该页
  - 存储开销为1/64=1.6%

另外一个重要的优化方向

- Merkle Tree的目的是防止replay attack，而CTR模式为每个block提供了一个counter
  - 如果我们能够保证counter的新鲜性（freshness），则可以保证数据的新鲜性
    - $MAC^{old} = H_k(Ctext^{old}, Counter^{old})$
    - Attacker replays MAC^{old} & $Ctext^{old}$ instead of MAC^{fresh} & $Ctext^{fresh}$
    - $MAC^{old} \neq H_k(Ctext^{old}, Counter^{fresh})$
      - Processor knows $Counter^{fresh}$
  - 因此，只需要使用Merkle Tree保护counter的新鲜性

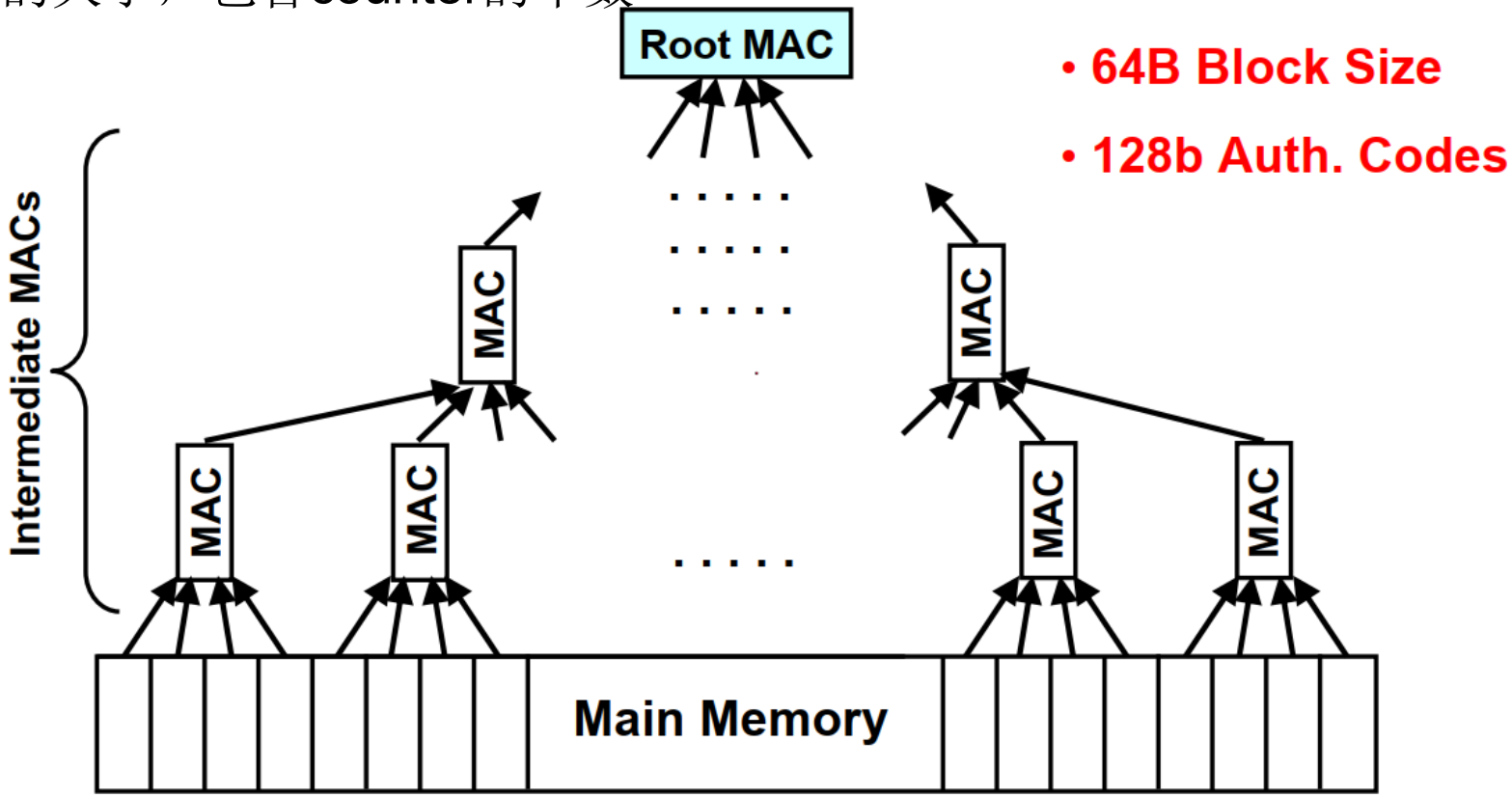
另外一个重要的优化方向

- counter要比data小的多（64B v.s. 4KB），因此一个Tree node可以容纳更多的counter
  - cache压力更小，查询更快



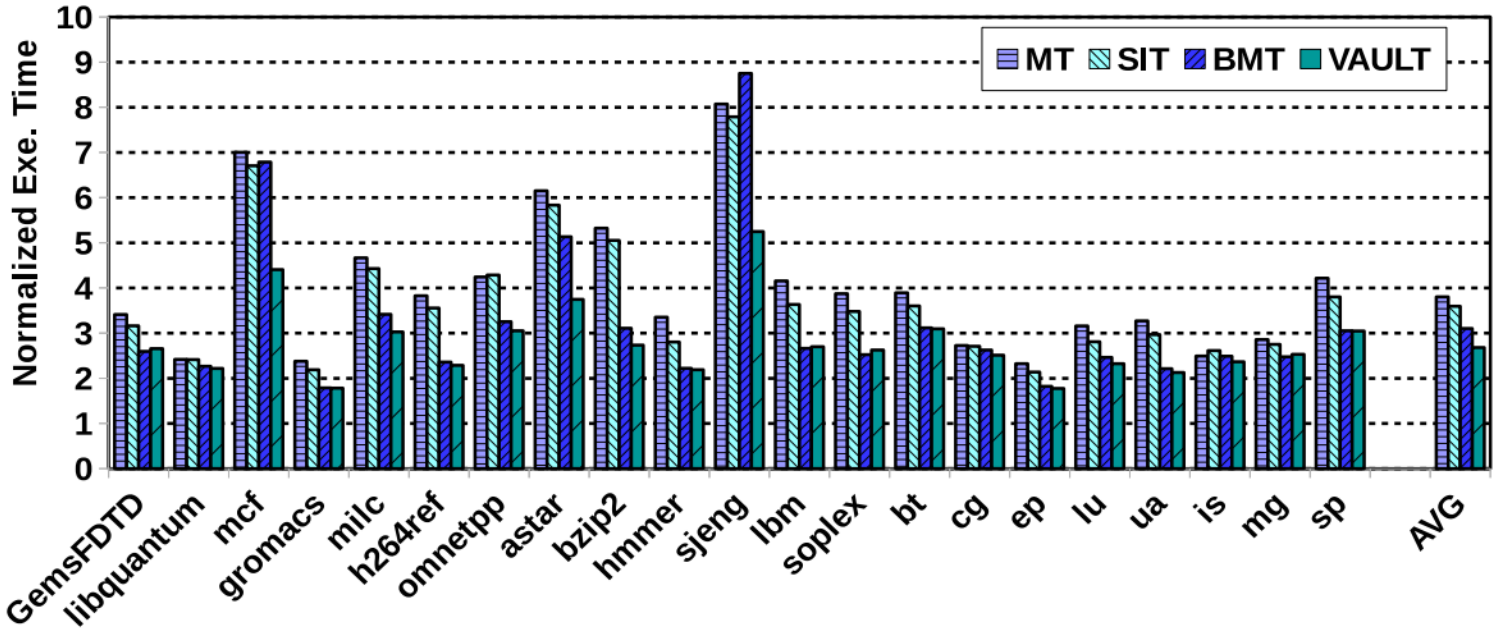
其它优化方向

- Merkle Tree的主要参数
  - 每一层的节点数
  - 树的深度
  - 每个节点的大小，包含counter的个数



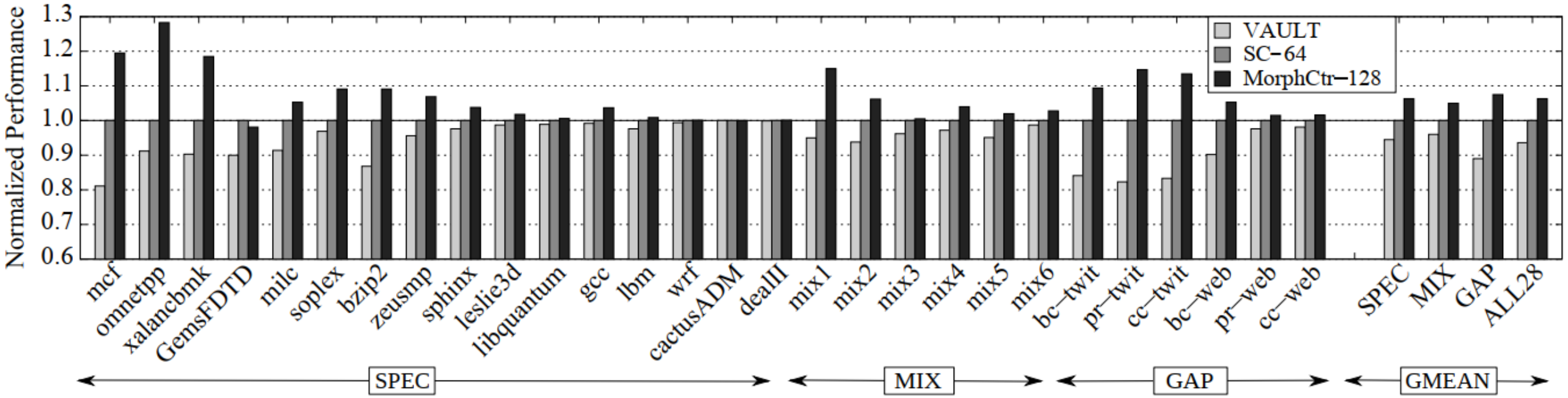
其他改进

- VAULT. ASPLOS 2018：增加可以支持的受保护内存大小
- EPC hit: 200 cycles; EPC miss 40K cycles
- SGX中树的每一层8个节点，VAULT使用了16到64个可变节点
- 压缩MAC的存储
- 16 GB受保护内存



其他改进

- Morphable Counters. MICRO 2018
  - 根据应用程序的特点，每个cache line中的counter数量是动态可变的（counter的大小可变，之前的方案都是固定的64个）
  - 通过对counter做压缩，降低Merkle tree的大小以及占用的cache大小，减少counter溢出后的重加密代价
  - 16GB受保护内存



其他改进

- Synergy. HPCA 2018
  - 将MAC放在ECC区域，对MAC的存取不需要额外的一次memory transaction
- Compact Leakage-Free Support for Integrity and Reliability. ISCA 2020
  - 对integrity tree中的metadata布局的优化，降低metadata miss的概率
- 蓬莱enclave
  - Mountable Merkel tree

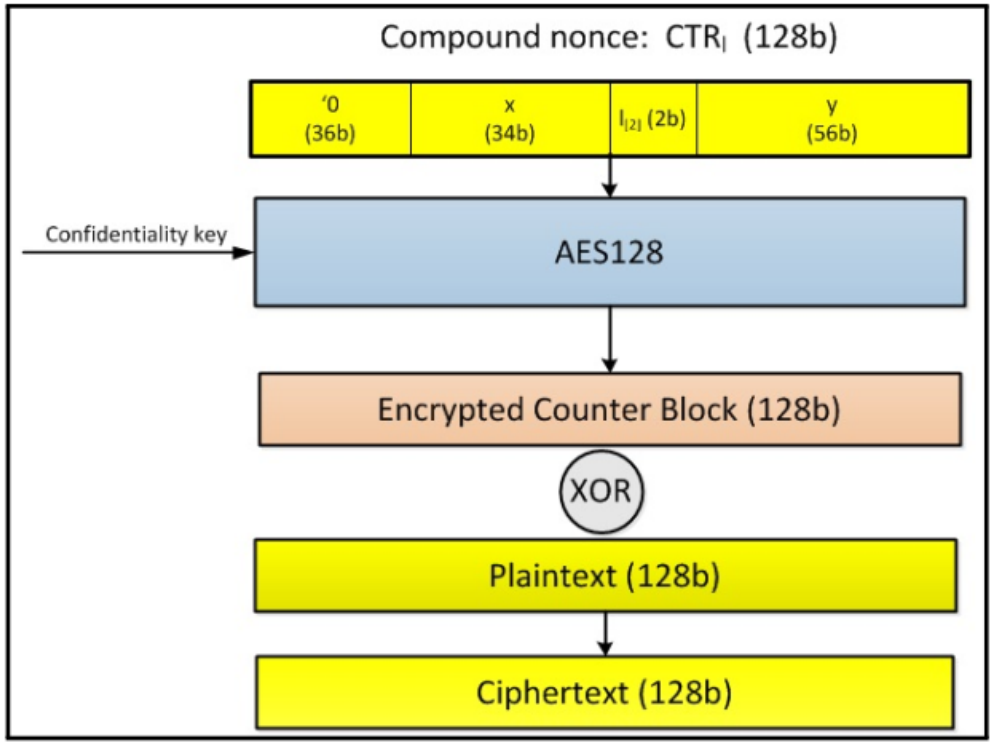
Intel SGX MEE

- 加密模式
  - x：spatial counter（40比特物理地址，对应34比特CL地址）
  - y：counter的范围（56比特）
  - 每个cache line分为4个AES分组（2个比特）

A Memory Encryption Engine Suitable for General Purpose Processors

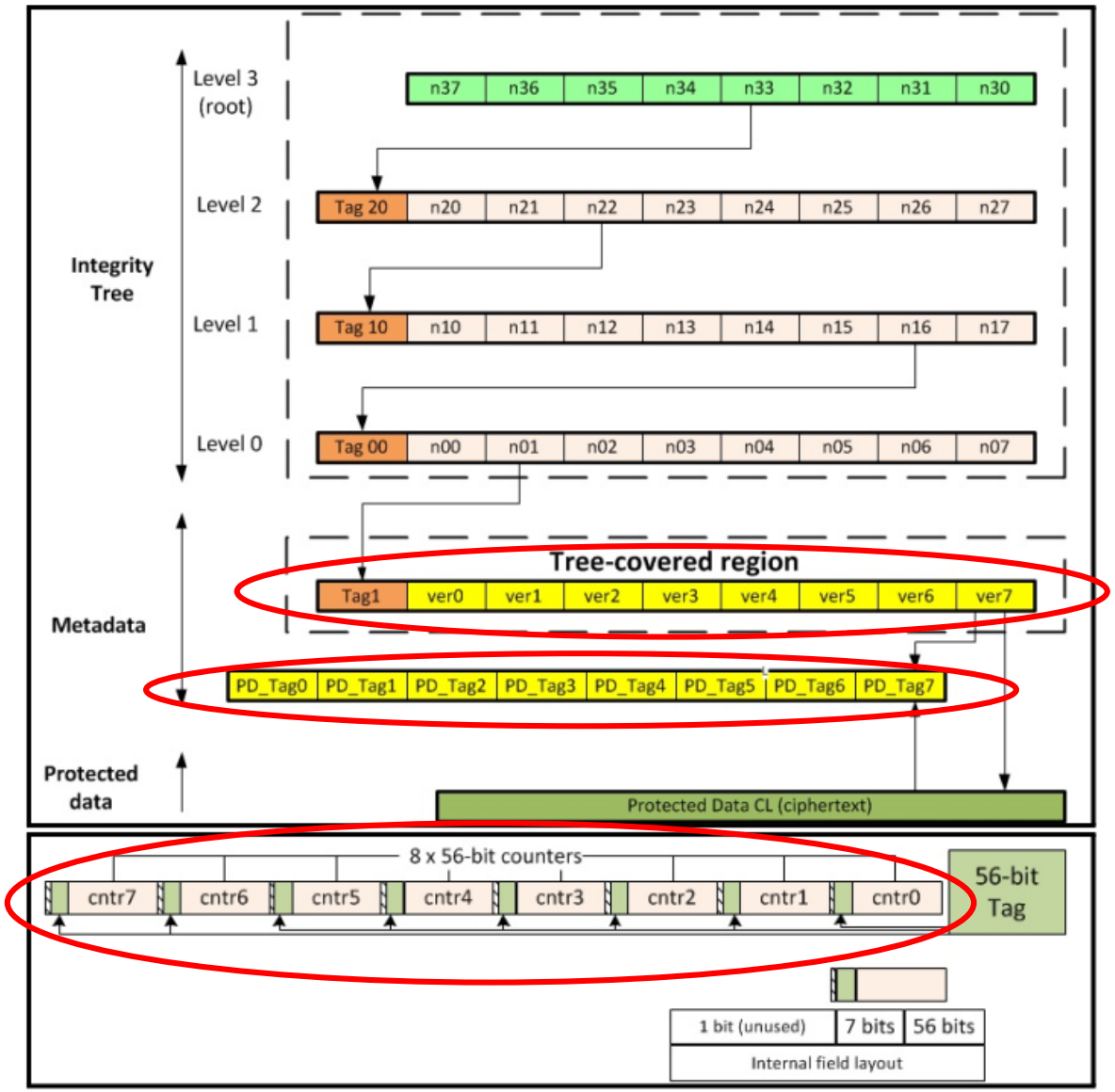
Shay Gueron[1,2]

[1]Intel Corporation, Intel Development Center, Israel
[2]University of Haifa, Israel



Intel SGX MEE

- 只对counter（图中的version）进行完整性保护
- Counter=address based spatial counter + temporal counter
  - 没有做counter压缩



SGX设定的可用范围为96 MB，tag和version都是数据的1/8，各12MB
自底向上，每一级为下一级的1/8

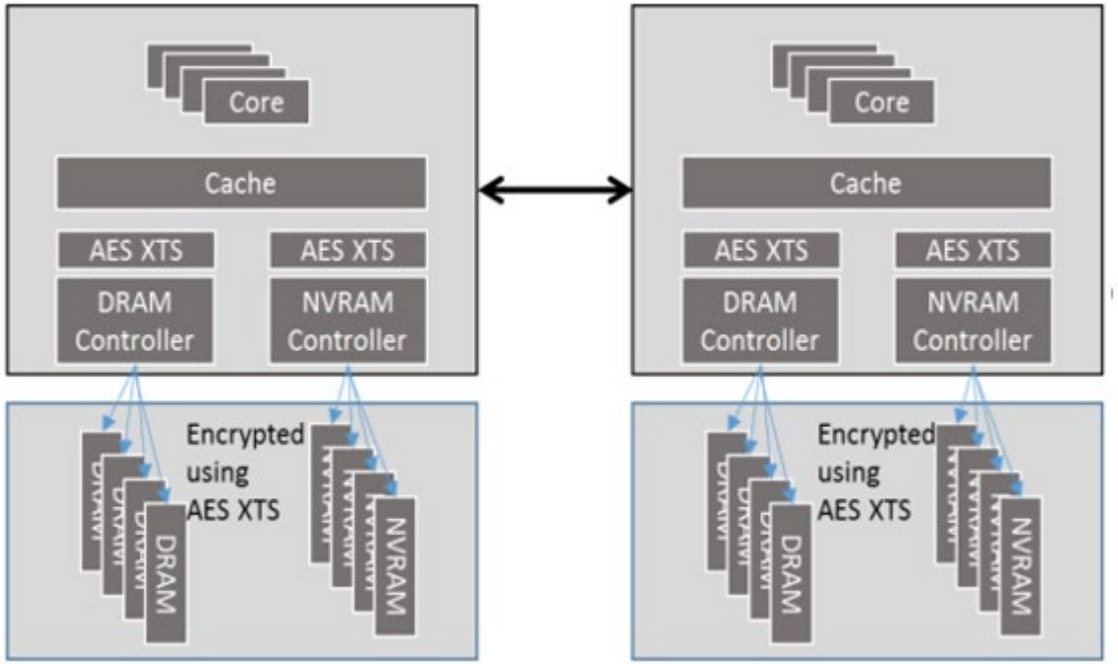
	Start offset	End offset	Region size
Protected data	000_0000	5ff_ffff	96MB
Version + PD_Tag	600_0000	77f_ffff	24MB
Reserved	780_0000	7df_ffff	6MB
L0	7e0_0000	7f7_ffff	1.5MB
Reserved	7f8_0000	7fb_ffff	256KB
L1	7fc_0000	7fe_ffff	192K
Reserved	7ff_0000	7ff_7fff	32KB
L2	7ff_8000	7ff_dfff	24K
Reserved	7ff_e000	7ff_efff	4KB
L3 (On-die root)	7ff_f000	7ff_ffff	4KB
Total MEE region			128MB

SGX V2/Scalable SGX

- 移除了基于Merkle Tree的完整性检查
- 利用ECC比特表明每个cache line是否属于某个enclave
- 降低了对片外存储的完整性保护
  - 无法防御memory bus replay attacks

Intel TDX/MKTME和AMD SEV系列

- AES XTS
  - 使用物理地址作为tweak
  - 同样的明文，在不同的物理地址，其密文是不同的



使用XTS加密模式

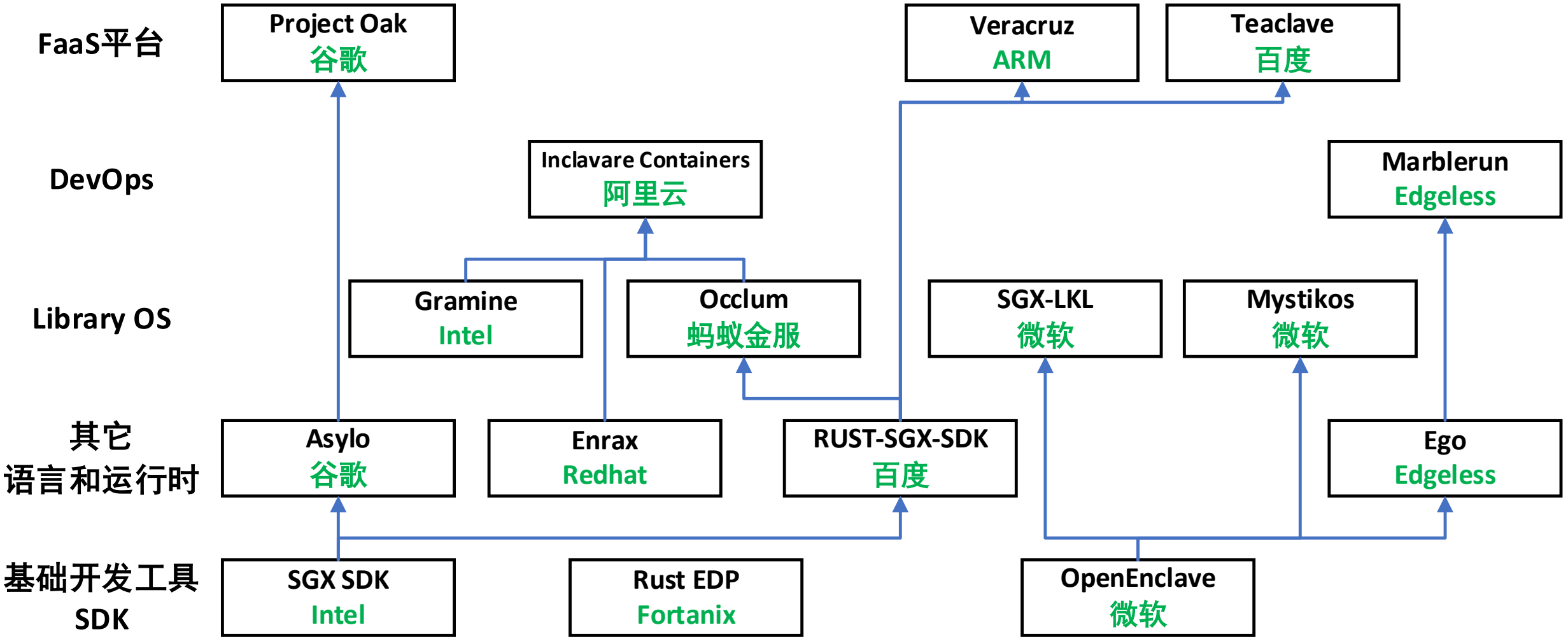
- SEV
  - 除加密外，不包含任何可以保护完整性的metadata
  - 不知道密钥的情况下的任何修改，会使得其变为随机值
    - 仍然可以被利用（*）
    - 仅适用于AMD SEV-SNP之前的版本
  - 逆向tweak后，可以使用已知的明文覆盖目标的物理地址
    - 仅适用于AMD SEV-SNP之前的版本
  - SEV-SNP防止了基于软件的完整性破坏
    - 但是无法防御基于物理的内存完整性攻击（包括bus replay attack）
  - 允许对加密数据进行读取，通过观察数据是否改变同样泄露信息
- TDX
  - 为每个64B自动生成28-bit的截断的sha-3输出作为MAC
  - MAC存储在DRAM ECC-bit之内，提供完整性保护
  - 1个额外比特用于表明cache line是否属于某个TD

总结

- TEE具有非常大的design space
  - 抽象层次
  - 威胁模型
    - 物理攻击、软件攻击、侧信道攻击
  - 隔离机制
- 安全性与性能、易用性的tradeoff
  - 适合的就是最好的
- TEE硬件设计不能解决所有问题
  - 软硬件协同架构

机密计算是一种新的计算范式

面向Intel SGX的软件栈



欢迎批评指正！

中国科学院信息工程研究所
INSTITUTE OF INFORMATION ENGINEERING,CAS