Protect Data of Virtual Machines with MKTME on KVM

Kai Huang @ Intel Corporation
kai.huang@intel.com

KVM Forum 2018

Intel Software
Intel OpenSource Technology Center

Legal Disclaimer

No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document.

Intel disclaims all express and implied warranties, including without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement, as well as any warranty arising from course of performance, course of dealing, or usage in trade.

This document contains information on products, services and/or processes in development. All information provided here is subject to change without notice. Contact your Intel representative to obtain the latest forecast, schedule, specifications and roadmaps.

The products and services described may contain defects or errors known as errata which may cause deviations from published specifications. Current characterized errata are available on request.

Copies of documents which have an order number and are referenced in this document may be obtained by calling 1-800-548-4725 or by visiting www.intel.com/design/literature.htm.

Intel and the Intel logo are trademarks of Intel Corporation or its subsidiaries in the U.S. and/or other countries.

*Other names and brands may be claimed as the property of others

Agenda

- Background & MKTME Introduction
- MKTME Use Cases
- MKTME Enabling & Status

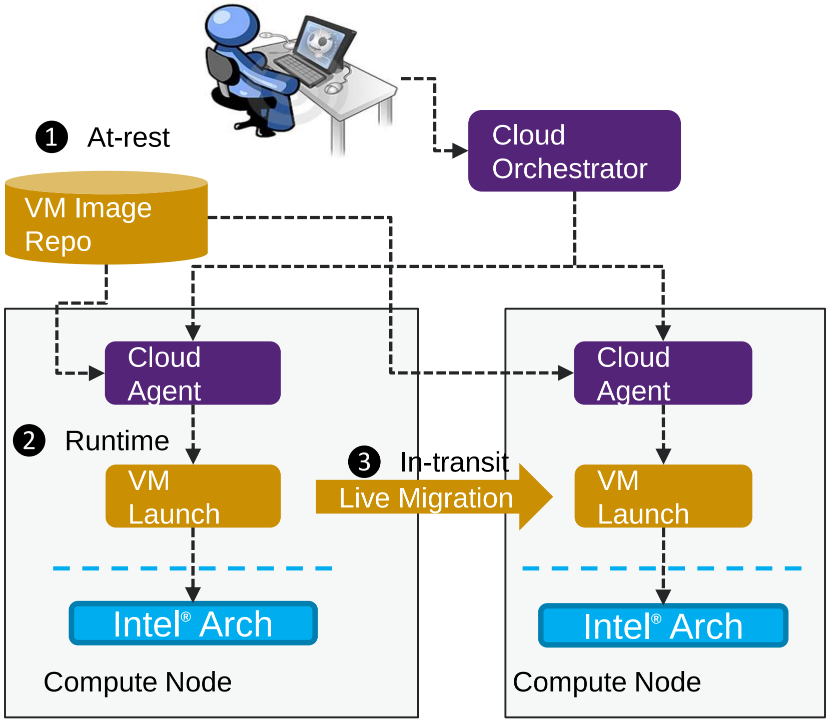
Background: Trusted VM in Cloud

VM protection by using encryption

- VM encrypted ‘at-rest’, ‘in-transit’ and ‘runtime’.
- There has been existing technologies for ‘at-rest’ and ‘in-transit’ encryption
  - Qemu TLS support for live migration
  - Qemu encrypted image support
- VM runtime encryption requires **hardware memory encryption** support
  - AMD® SME/SEV
  - Intel® MKTME

Launch VM on ‘Trustiness Verified’ Host

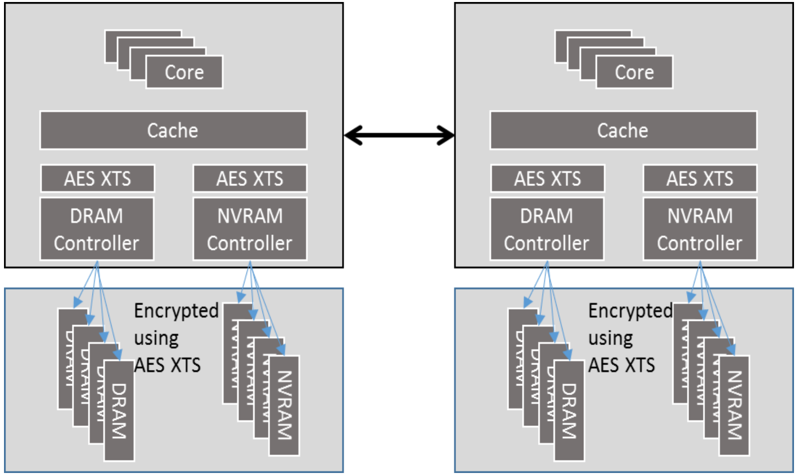
- Trusted hardware, SW stack, etc.
  - HW based root-of-trust
  - Attestation service



Typical VM Lifecycle in Cloud

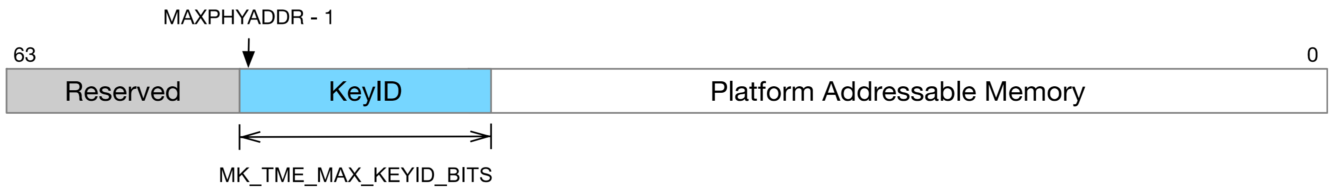
TME & MKTME Introduction

- New AES-XTS engine in data path to external memory bus.
  - Data encrypted/decrypted on-the-fly when entering/leaving memory.
  - AES-XTS uses physical address as “tweak”
    - Same plaintext, different physical address -> different ciphertext.
- TME (Total Memory Encryption)
  - Full memory encryption by TME key (CPU generated).
  - Enabled/Disabled by BIOS.
  - Transparent to OS & user apps.
- MKTME (Multi-key Total Memory Encryption)
  - Memory encryption supporting using multiple keys.
  - Use upper bits of physical address as keyID (see next)



MKTME KeyIDs

- Repurpose upper bits of physical address as KeyID as shown below.
  - Reduces useable physical address bits.
  - Different keyIDs can refer to the same physical address.
- Architecturally upto 2^15-1 keyIDs (15 keyID bits).
  - Reported by MSR. Configured by BIOS.
  - KeyID 0 is reserved as TME's key (not useable by MKTME).
- New PCONFIG instruction to program keyID w/ associated key (see next)



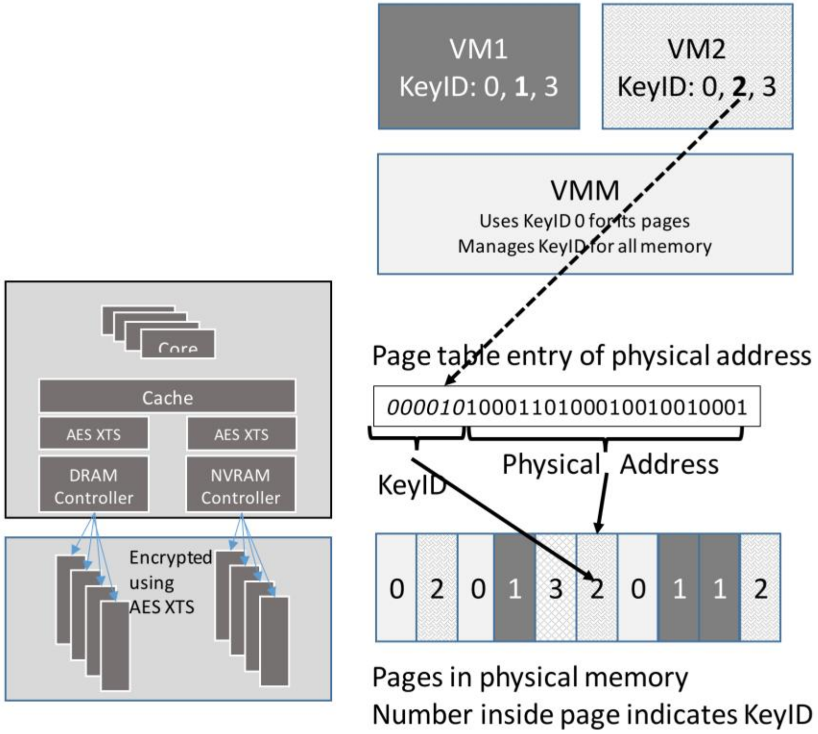
MKTME KeyID Programming Overview

New Ring-0 instruction PCONFIG to program the KEYID and associated key

- Package scoped
- Supports programming keyID to 4 modes:
  - Using CPU generated random ephemeral key (invisible to SW)
    - SW can provide entropies for key and tweak, which will be XOR-ed by CPU.
  - Using SW provided key (tenant’s key)
  - No encryption – plaintext domain
  - Clearing a key (using TME’s key effectively)
- Allows SW to specify crypto algorithms
  - Only AES-XTS-128 for initial server intercept

VM Protection & Isolation With MKTME

- Protection
  - Use keyID to encrypt VM memory at runtime
- Isolation
  - Use different keyIDs for different VMs
- Software Enabling
  - For CPU access, SW sets keyID at PTEs
    - IA page table (host)
    - EPT (KVM)
  - For Device access (DMA)
    - w/ IOMMU: Set keyID to IOMMU page table
    - Physical DMA: Apply keyID to PA directly



Recap -- Highlights of MKTME

Guests continue to run "*without modifications*" in MKTME guest:

- Encrypted with 1) CPU-generated ephemeral key, or 2) the one provided by API ("tenant-controlled keys")
- Virtio, including optimization (direct access to guest memory by kernel) continues to work
- Direct I/O (including accelerators, FPGA) assignment (including SR-IOV VFs) is available
- Live migration can be supported (among platforms that support MKTME)
- vNVDIMM can be supported w/ limitation (because of physical address "tweak")
  - Host DIMM configuration cannot be changed cross reboots.
  - Qemu DIMM & vNVDIMM configuration cannot be changed cross VM reboots.

Agenda

- Background & MKTME Introduction
- MKTME Use Cases
- MKTME Enabling & Status

MKTME Enabled Use Cases

1. ***Launch Tenant VMs with runtime protection with CPU generated keys***

- Let CSP handle the keys
- VM image provided by CSP

2. ***Launch Tenant VMs with at-rest and runtime protection with full tenant-control keys***

- VM image encrypted at-rest with tenant-specific keys
- VM memory isolation with tenant-specific keys
- Keys fully controlled by tenant
- Trustiness verified host
- Additional: integrity verification of VM image

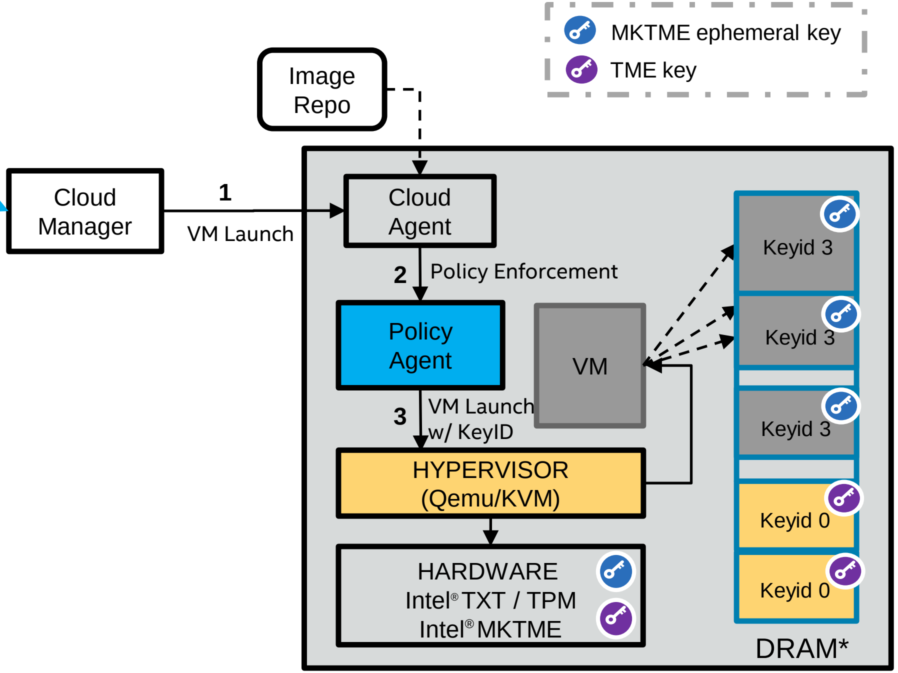
Use-case Extension:

KeyID Sharing for all VMs launched by single tenant with the same tenant-key (or CPU generated key).

VM Launch w/ CPU Generated Keys

VM Launch w/
- CPU generated key
- CSP provided VM image

Security Properties
- w/ VM runtime protection
- w or w/o at-rest and in-transit protection
- No Host Trustiness Verification



CSP Controlled

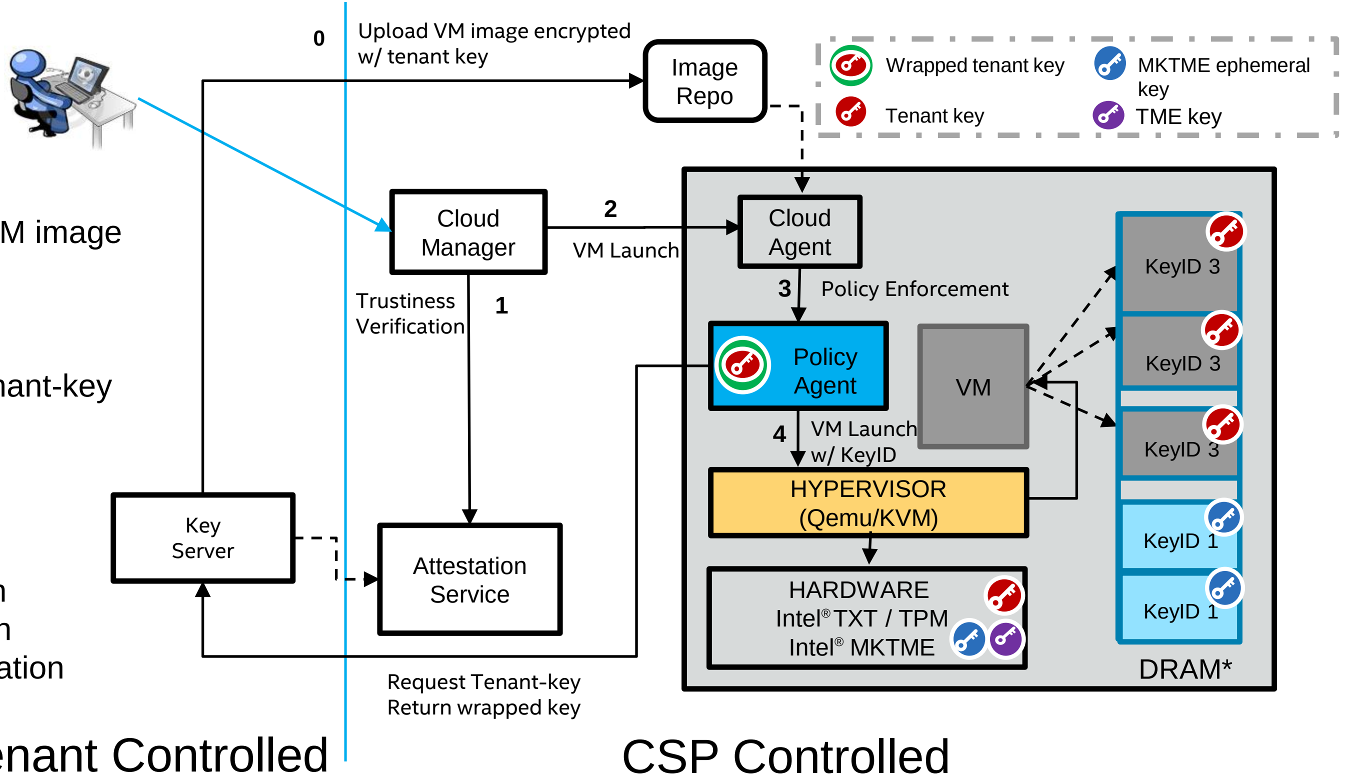
VM Launch w/ Tenant Controlled Keys

VM Launch w/

- Tenant provided key
- Tenant provided encrypted VM image
- Tenant controlled key server
- Trustiness verified host
- VM image integrity verified
- Use TPM to wrap/unwrap tenant-key

Security Properties

- w/ VM runtime protection
- w/ VM at-rest protection
- w/ or w/o in-transit protection
- w/ Host trustiness verification
- w/ VM image integrity verification



KeyID Sharing Among VMs

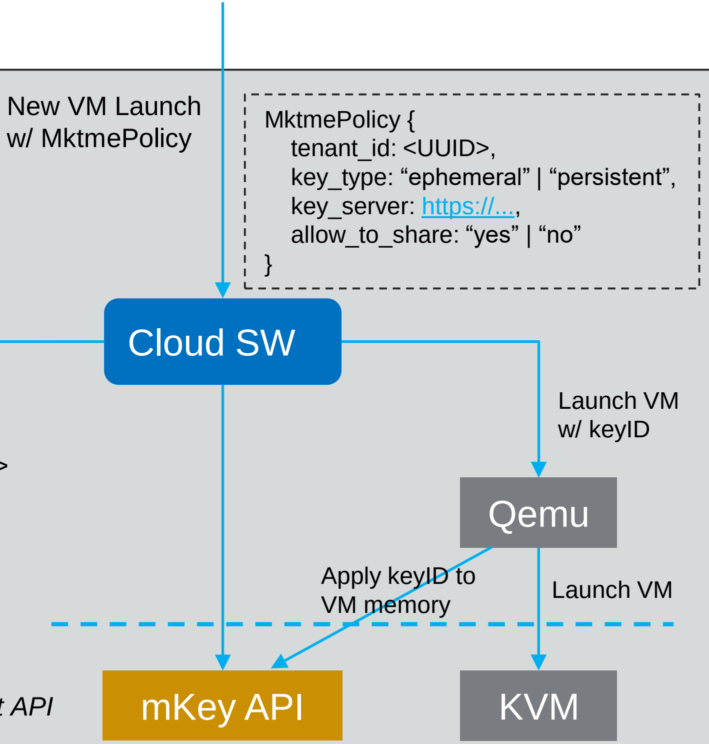
Cloud SW makes decision whether to share or not.

KeyIDPolicy	KeyID	VMs
Policy1: <tenant1, “ephemeral”>	keyID1	VM1, VM2..
Policy2: <tenant2, “persistent”, xxxxxx>	keyID2	VM3

Example: KeyID sharing is based on KeyIDPolicy: <tenant_id, key_type, tenant_key>

Cloud SW:

- *Maintains ‘KeyIDPolicy-to-KeyID’ table*
- *Makes keyID sharing decision according to the table*
- *Updates the table on VM launch and teardown*



Compute Node

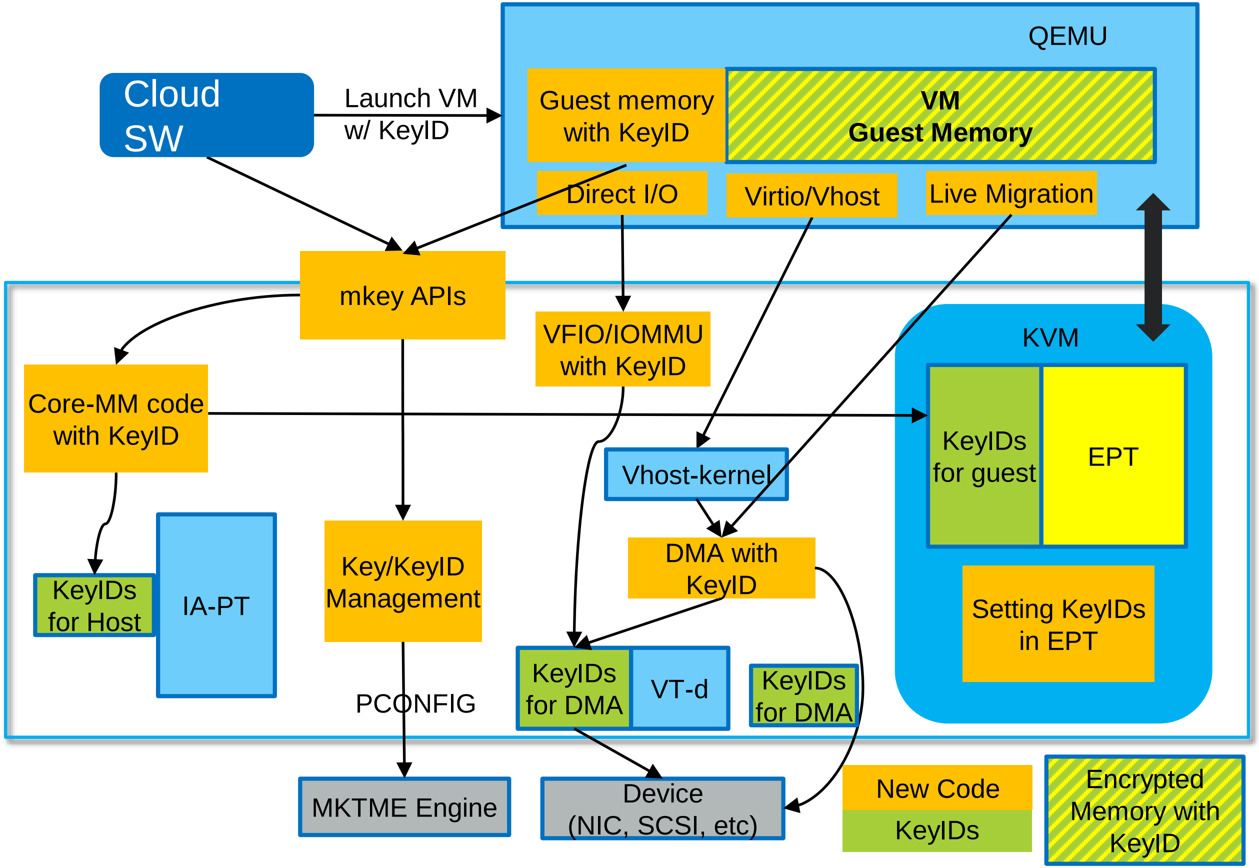
mKey API: MKTME key management API

Agenda

- Background & MKTME Introduction
- MKTME Use Cases
- MKTME Enabling & Status

MKTME Enabling Overview

- **Overall:**
  - Setup the same keyID in both Qemu host page table and EPT/shadow page table
- **Passthrough:**
  - Setup keyID to IOMMU
- **Virtio/vhost-kernel:**
  - kmap() w/ keyID
  - DMA w/ keyID
- **Live Migration:**
  - DMA w/ keyID



MKTME Enabling Overview -- Recap

- **Host kernel**
  - Key/keyID Management APIs
    - Use kernel key retention services infrastructure, and add new MKTME key type.
    - Return ‘key_serial_t’ (handle) to userspace instead of actual keyID.
  - Core-MM keyID support
    - VMA, page table keyID manipulation
    - Setup keyID to PTE in #PF
  - New syscall to encrypt process memory region by given MKTME key handle.
    - encrypt_mprotect(addr, size, prot, key_handle);
  - VFIO/IOMMU keyID support, DMA keyID support.
- **KVM**
  - Setup keyID in EPT/Shadow MMU
- **Qemu**
  - Receive MKTME key handle from Cloud SW
  - Apply MKTME key handle to all guest memory (by calling new syscall)

MKTME Enabling – Qemu Modification

- New “mktme-guest” object to carry MKTME handle
  - -object mktme-guest,id=mk0,handle=$mktme-handle
  - Align with AMD SEV’s “sev-guest” object
- Reuse machine property “memory-encryption” to indicate VM is associated w/ keyID.
  - Consistent with AMD SEV, who introduced ‘memory-encryption’ property

Example: Launch VM w/ $mktme-handle

```
#qemu-system-x86_64 … -machine memory-encryption=mk0 -object mktme-guest,id=mk0,handle=$mktme-handle
```

Example: Put into a small script, combined w/ adding MKTME key

```
#!/bin/bash
serial=`keyctl add mktme k1 “type=cpu algorithm=aes-xts-128” @us`
qemu-system-x86_64 –enable-kvm –cpu host –smp 2 –m 4G –machine memory-encryption=mk0 \
        -object mktme-guest,id=mk0,handle=${serial}
```

MKTME Enabling Current Status

- Specification has been published [1]
- Core kernel enabling status
  - Some preliminary patches have been upstreamed
    - Feature emulation (CPUID, MSR); PCONFIG
  - Some RFCs have been sent to community for feedback
    - New MKTME key type implementation
  - Other components WIP internally
    - Core-MM keyID support; IOMMU keyID support; DMA keyID support; …
- KVM/Qemu enabling status
  - PoC has been done to prove MKTME actually works.
  - Depending on core kernel parts ready for formal patches.

[1] https://software.intel.com/sites/default/files/managed/a5/16/Multi-Key-Total-Memory-Encryption-Spec.pdf

THANKS

Intel Software

INTEL OpenSource TECHNOLOGY CENTER