Flash Memory Summit

Key Per IO - Fine Grain Encryption For Storage

With TCG's Per-I/O Encryption Key Selection

Frederick Knight, NetApp

35 YEARS OF NAND FLASH MEMORY INNOVATION

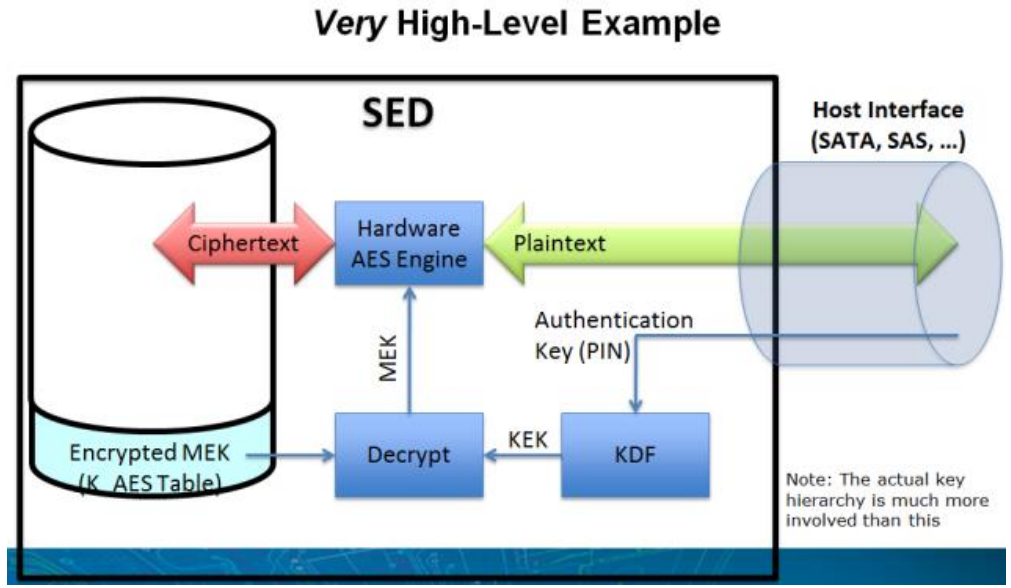
Agenda

- Data at Rest Protection - Background
- Key Per I/O - Goals
- Key Per I/O - Key Flow
- Key Per I/O - SSC and I/O Architecture Interactions

Existing Data At Rest Protection vs. Key Per IO

Background On Data At Rest Protection

Data At Rest Protection



Properties

- Encrypt all user accessible data all the time, at interface speeds
- Keys generated & stored in NVM by the storage device
- Media Encryption Key (MEK) associated with contiguous LBA ranges or Namespaces
- Opal/Enterprise SSC* deliver passwords to drive in the clear (when not using Trusted Computing Group (TCG)* - Secure Messaging)

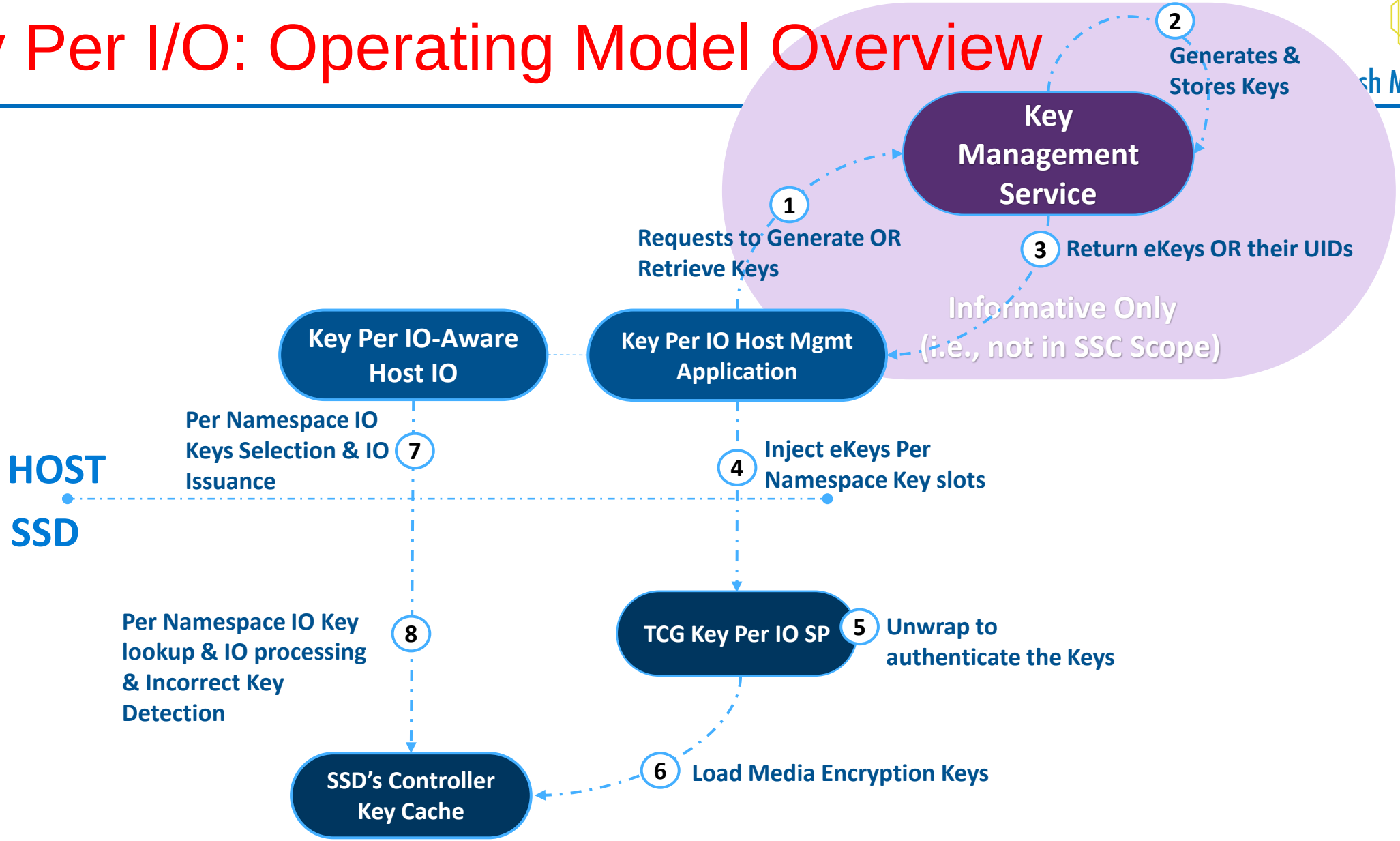
*Other names and brands may be claimed as the property of others.

Can we do better?

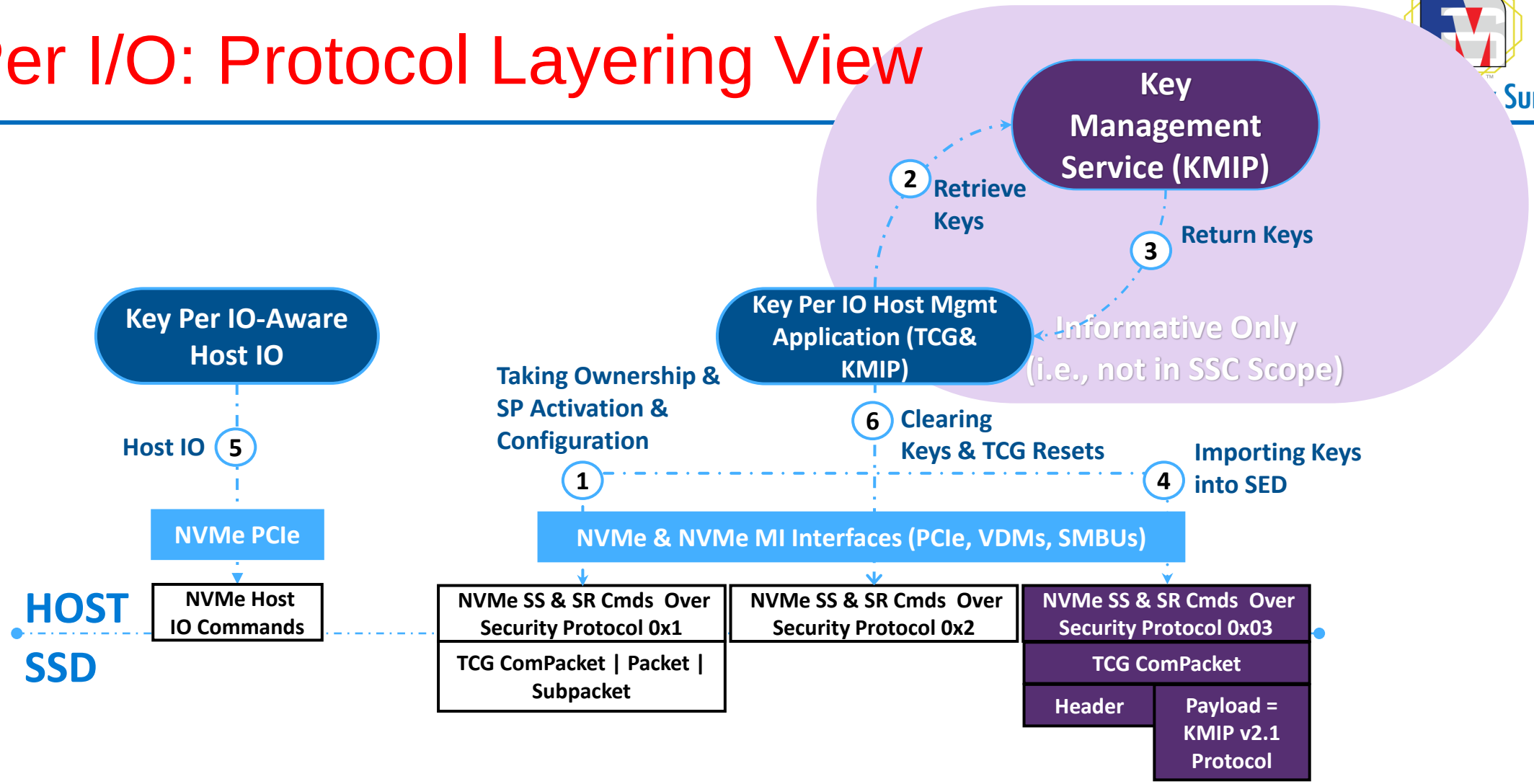
Desired properties:

1. Select an encryption key for each I/O to a Storage Device?
    - Associate encryption domains with higher-level objects (abstractions) than drives or volumes.
    - Crypto erase individual higher-level objects
    - Easier to support European Union's General Data Protection Regulations' "Right to be forgotten"
2. Externally manage Media Encryption keys?
    - Centralized key management infrastructure, consistent key policies
    - High assurance key generation and control, e.g., master keys in HSM (Hardware Security Module)
3. Ensure that a Storage Device with no power has no encryption keys?
    - Shorter physical drive loss/theft discussion with security auditor
    - Easier decommissioning process

Key Per I/O: Operating Model Overview



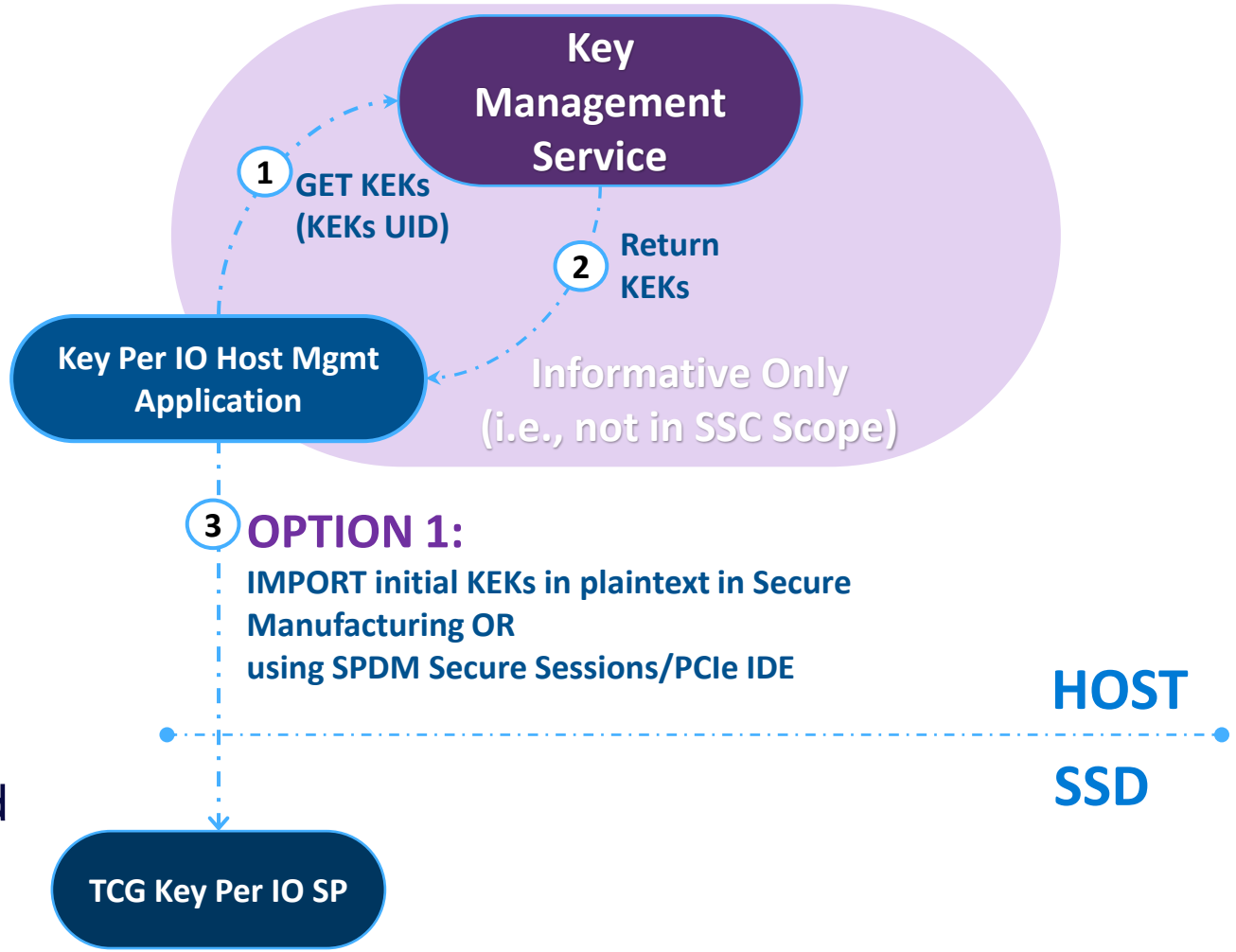
Key Per I/O: Protocol Layering View



Key Per I/O: Injecting Keys

KEKs Transport Security:

- Objectives:
  - Confidentiality
    - **Option 1**:
      Initial KEKs are pre-shared in secure manufacturing or over secure transport link (e.g., SPDM Secure session, PCIe IDE)
  - Authentication & integrity
    - Subsequent KEKs are transported wrapped via AES-GCM or NIST AES-KeyWrap using initial KEKs

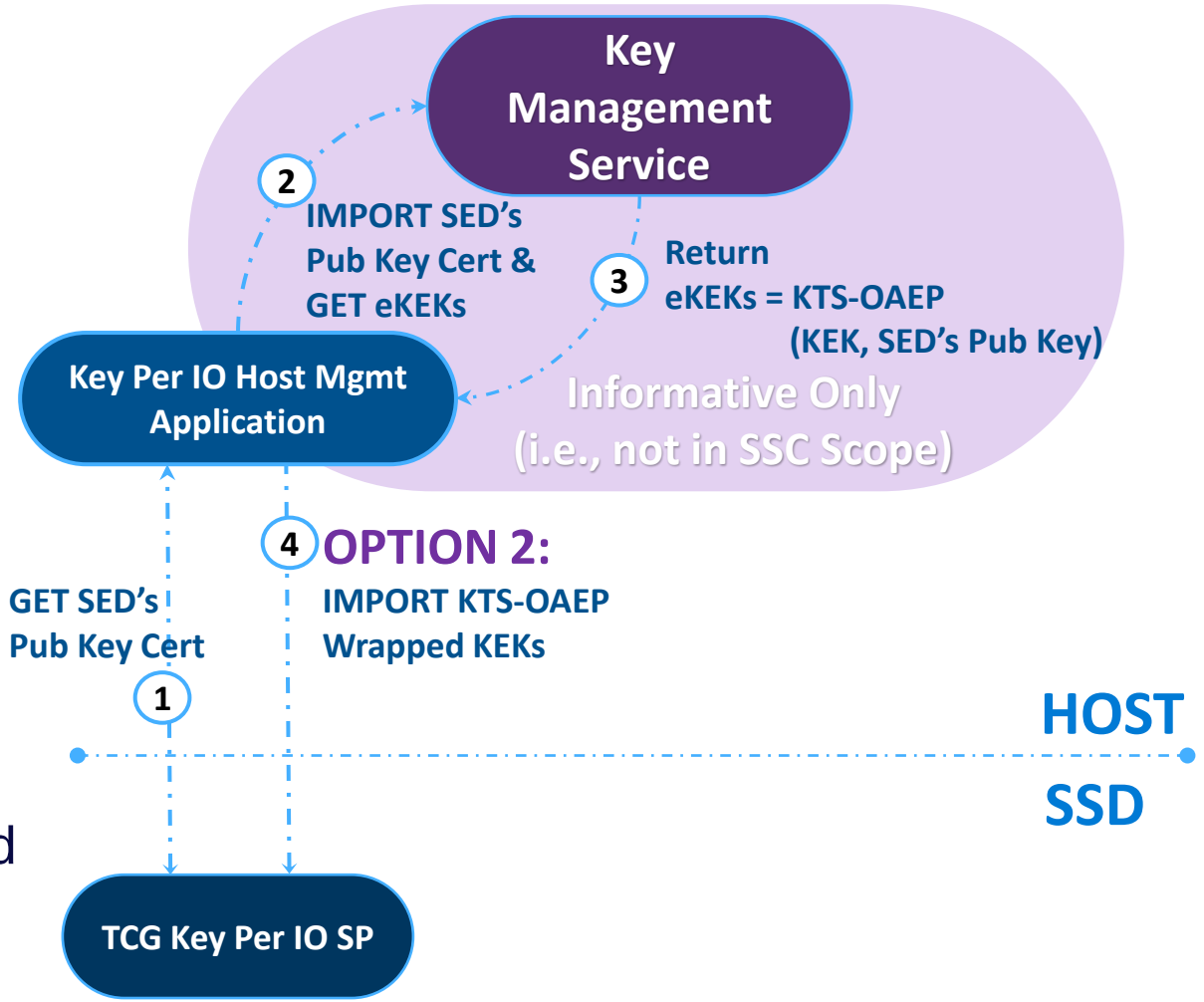


Key Per I/O: Injecting Keys

Flash Memory Summit

KEKs Transport Security:

- Objectives:
  - Confidentiality
    - **Option 2**:
      Initial KEKs are pre-shared using KTS-OAEP & SED's Pre-provisioned
      Key Per IO Public Key Certificate
  - Authentication & integrity
    - Subsequent KEKs are transported wrapped via AES-GCM or NIST AES-KeyWrap using initial KEKs

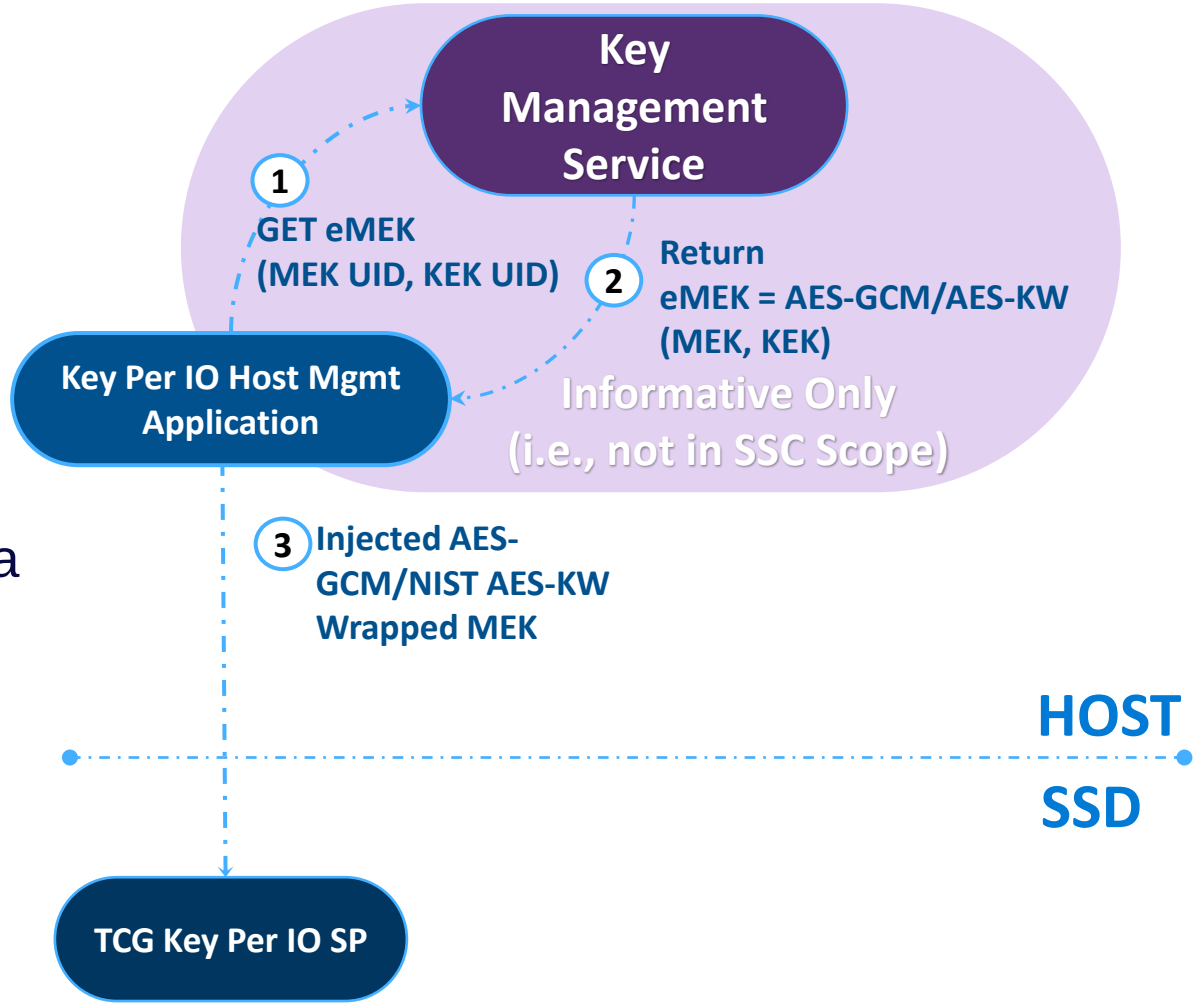


35 YEARS OF NAND FLASH MEMORY INNOVATION

Key Per I/O: Injecting Keys

MEKs Transport Security:

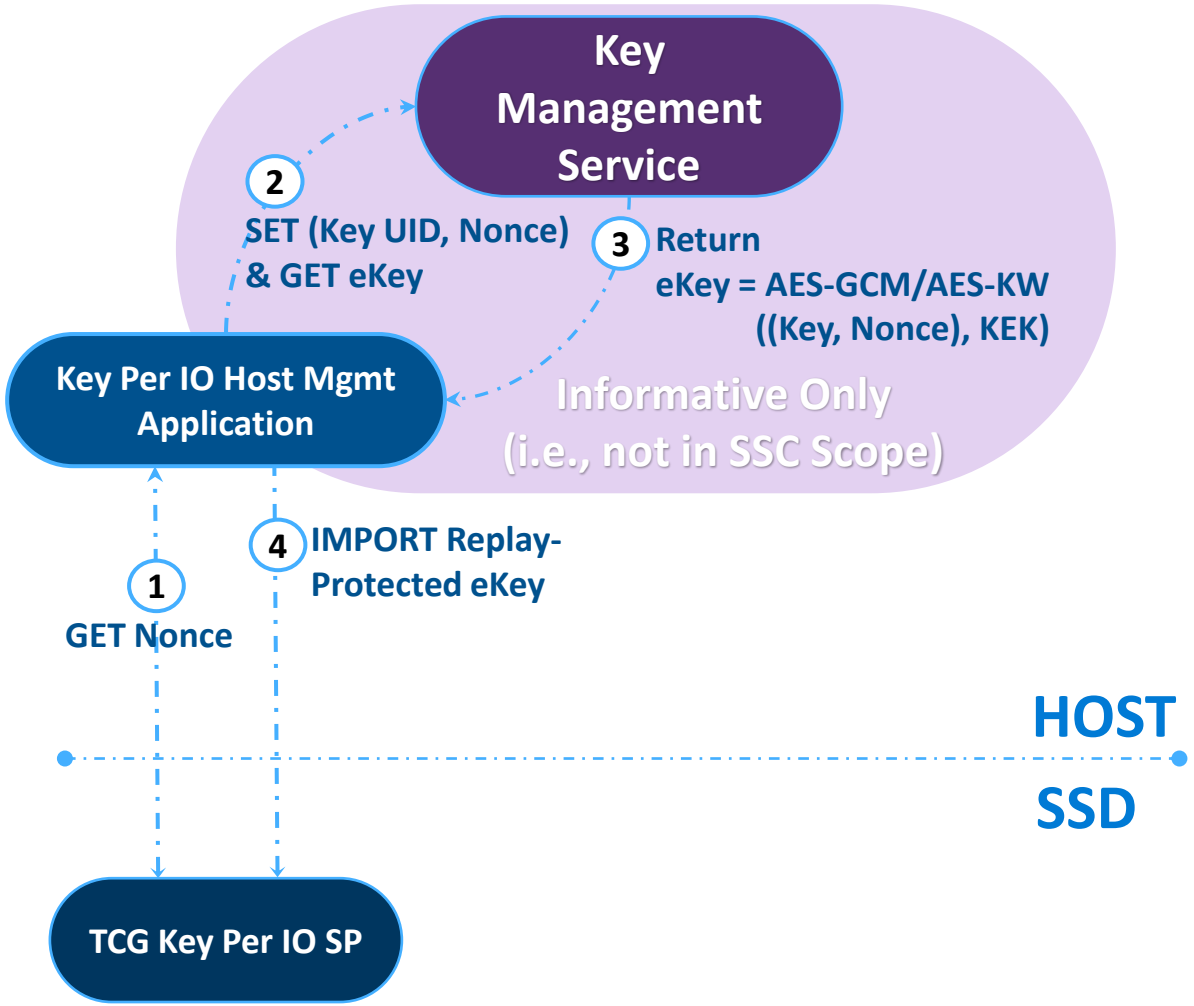
- Objectives:
  - Confidentiality, Authentication & integrity
    - MEKs are transported wrapped via AES-GCM or NIST AES-KeyWrap using pre-shared KEKs



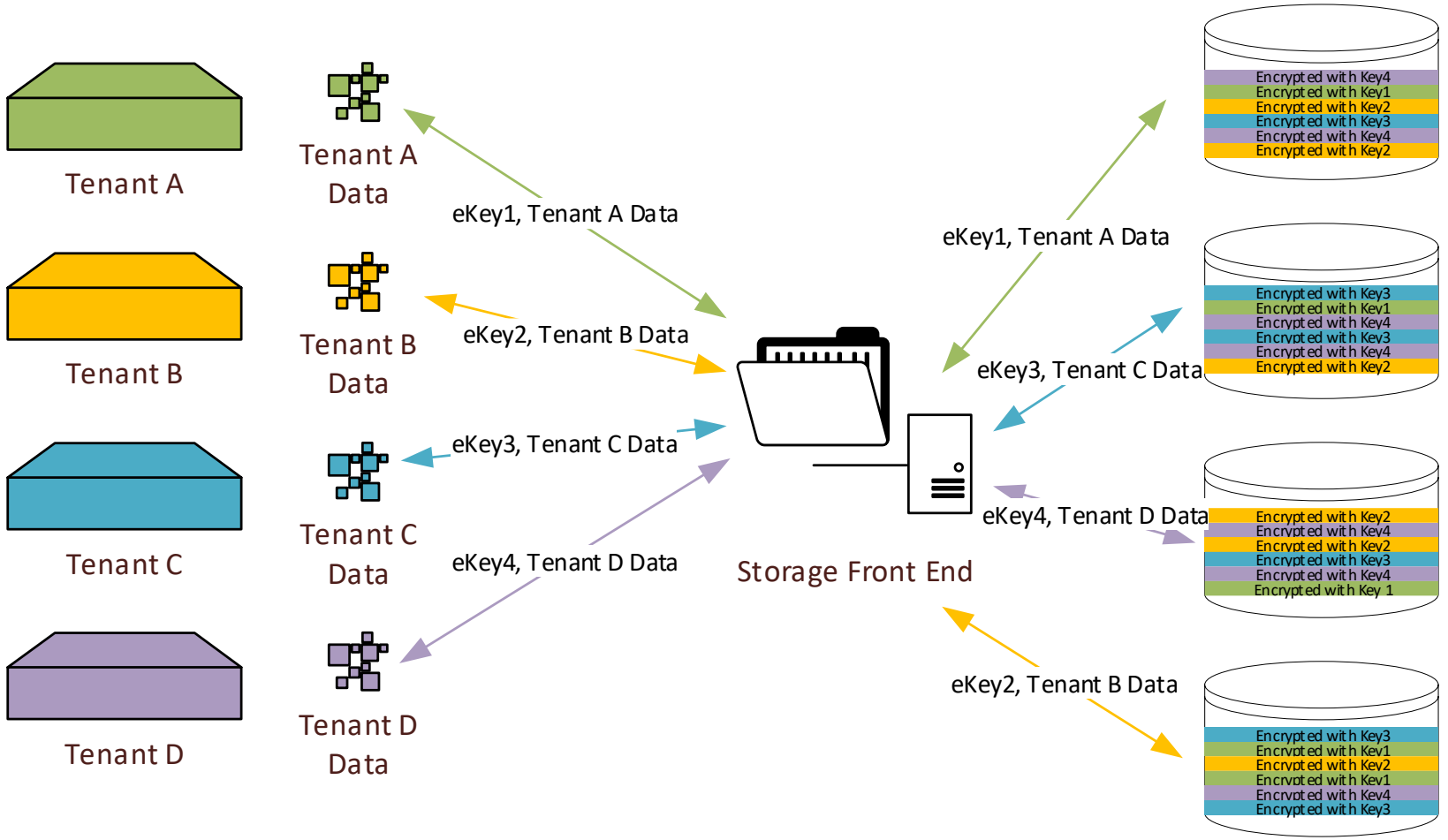
Key Per I/O: Injecting Keys

Replay Protection:

- Objective:
  - Preventing replay of old copies of keys by unauthorized entity
    - Nonce is cryptographically tied to key during injection.
    - Encryption authentication tag has to encompass both key & Nonce



Key Per I/O:
Example Use Case – Tenant Isolation



Key Per I/O SSC and I/O Architecture Interactions

KPIO Discovery

Host Detection of KPIO

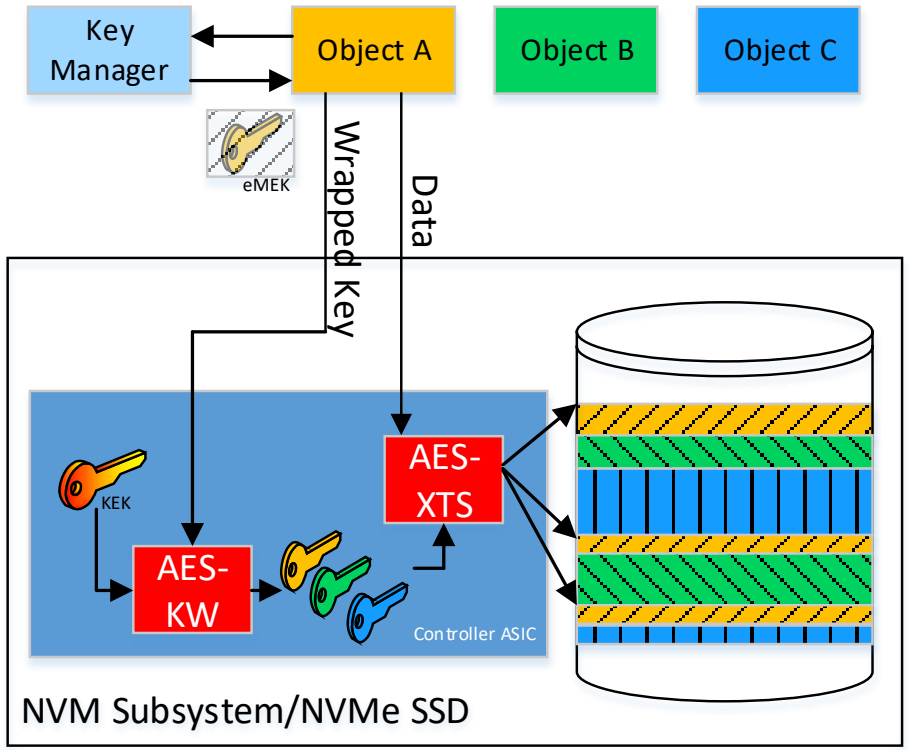
- Number of Key Tags supported
- Granularity and alignment of operations
- NVMe Identify command
  - Per namespace
- TCG Discovery (Security Send and Security Receive)
  - Authenticate
  - Security Receive (Level 0 Discovery)
  - Discovery security characteristics

KPIO Configuration

Establish KEKs (Key Encryption Keys)

- Agreement between host and device for secure transmission of the KEKs (secure manufacturing (pre-shared keys), public/private key pairs (PKI), certificate, etc)
- Host obtains MEK (Media Encryption Keys) from a key management database (e.g., KMIP)
- MEKs are “wrapped” with KEKs and sent to the device

Wrapped MEKs sent from the host to the device



KPIO Configuration

MEKs are Loaded the Device Key Cache

- Associate each MEK injected with a Key Tag
  - Per namespace – loaded using Security Send command

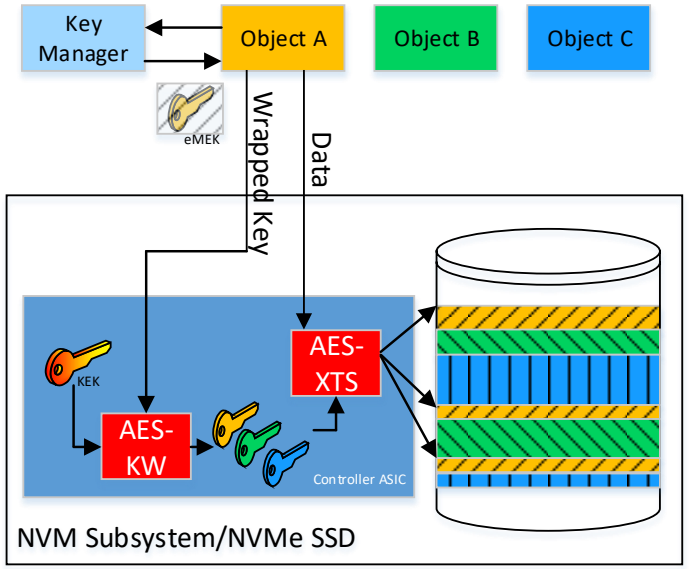
Key Tag	MEK example (256 bit)
1	0x1234567890ABC…90ABCDEF
2	0x1234567890ABC…90ABCDE0
100	0x1234567890ABC…90ABCDE1
101	0x1234567890ABC…90ABCDE2
103	0x1234567890ABC…90ABCDE3
200	0x1234567890ABC…90ABCDE4
217	0x1234567890ABC…90ABCDE5

KPIO Configuration

Load New Keys

- Associate a Key Tag with a different MEK
  - Per namespace – loaded using Security Send command

Key Tag	MEK example (256 bit)
1	0x1234567890ABC…90ABCDEF
2	0x1234567890ABC…90ABCDE0
100	0x1234567890ABC…90ABCDE6
109	0x1234567890ABC…90ABCDE7
110	0x1234567890ABC…90ABCDE8
111	0x1234567890ABC…90ABCDE9
220	0x1234567890ABC…90ABCDEA



KPIO Usage

I/O Command Usage

- Compare
- Copy
- Verify
- Read
- Write
- Write Zeroes
- Zone Append

- A field in each command to specify the Key Tag value to use for that individual I/O
- An indicator that a Key Tag is present

Key Tag	MEK example (256 bit)
1	0x1234567890ABC…90ABCDEF
2	0x1234567890ABC…90ABCDE0
100	0x1234567890ABC…90ABCDE6
109	0x1234567890ABC…90ABCDE7
110	0x1234567890ABC…90ABCDE8
111	0x1234567890ABC…90ABCDE9
220	0x1234567890ABC…90ABCDEA

KPIO Example Commands

- WRITE (LBA=100, LEN=8, flag=1, keytag=1)

MEK = 0x1234567890ABC…90ABCDEF

- WRITE (LBA=200, LEN=16, flag=1, keytag=100)

MEK = 0x1234567890ABC…90ABCDE6

- READ (LBA=100, LEN=8, flag=1, keytag=1)
  - Gets your data back
- READ (LBA=200, LEN=16, flag=1, keytag=1)
  - Gets error or bogus data
- READ (LBA=200, LEN=16, flag=1, keytag=100)
  - Gets your data back

Flag = 1 means the keytag is present

Key Tag	MEK example (256 bit)
1	0x1234567890ABC…90ABCDEF
2	0x1234567890ABC…90ABCDE0
100	0x1234567890ABC…90ABCDE6
109	0x1234567890ABC…90ABCDE7
110	0x1234567890ABC…90ABCDE8
111	0x1234567890ABC…90ABCDE9
220	0x1234567890ABC…90ABCDEA

KPIO Impact in TCG

Security Send / Security Receive Commands

- Uses new TCG protocol ID (0x03)
- Authentication
- Discovery
- Key Injection method (Establish Key Tag to MEK association)
- Key Clear method (Remove Key Tag to MEK association)
- Key Replacement method (Replace MEK for a Key Tag)
- Securely Purge Key Cache
- Define encryption / decryption algorithms that can be supported (e.g., XTS-AES-256)

KPIO Impact On Hosts

Host Responsibilities to use KPIO

- Hosts must manage the full life cycle of the Keys
  - Including secure purging of the keys
- Host is responsible for the correctness of the MEK injection / key tag association and use of the correct key tag for each I/O command
- Host is responsible for preventing incorrect key tag use
  - Key tag associations may change during operation – such as key tag cache size smaller than key tag needed usage
  - Using the key tag associated with the correct MEK
- Host must handle errors for improper use of key tags
  - Invalid key tag value (out of range), or a key tag with no associated MEK
  - Trying to use a key tag before injection is complete or after removal

KPIO Project Status

Current Key Topics in Progress

- Details of the MEK / KEK secure injection process almost complete
  - KMIP based methods
- Incorrect MEK detection optional capability
  - Incorrect MEK should not look like a Media Error
  - Does incorrect MEK just return “bogus” data
  - UUID association
- Testing use cases
- Still a work in Process – but almost finished

- NVMe work is nearly complete

https://nvmexpress.org/

- TCG work is almost complete
- Come join us at TCG Storage Work Group to continue the discussions!

https://trustedcomputinggroup.org/

KPIO For Other IO Architectures

What about SCSI and/or SATA

- The same TCG architecture is used by SCSI and SATA
  - Security Send / Security Protocol Out / Security Receive / Security Protocol In
- But completely new I/O commands would be required
  - Such as 32-byte CDBs for SCSI (to carry the Key Tag value)

- NO interest being shown to undertake such an effort

KPIO Key Takeaways

- The KPIO SSC is being defined such that an SD that claims TCG Opal SSC compatibility could be a KPIO SSC.
- Intended to protect confidentiality of data at rest from unauthorized access once it leaves the owner's control.
- Creating a fine-grained approach to enhance SED technology to better support multi-tenancy usage models.
- Standards based designs for multi-vendor interoperability.

Please don’t forget to rate this session.

- Your feedback is important to us.

