*presented by*

# UEFI Key Management Service (KMS) With TPM

UEFI Fall 2023 Developers Conference & Plugfest

October 9-12, 2023

Presented by Felix Polyudov and Frederick Otumfuor

# Agenda

- Introduction
- TPM-based KMS Design
- KMS Improvements
- Questions

# What and Why?

- What's this presentation about?
  - UEFI Key Management Service (KMS)
    - Theory of operation
    - Implementation Options
    - Introduce a TPM-based solution
- Why KMS?
  - Educational value: one of the less known UEFI protocols
  - Practical value: helps solving real-life use cases
    - Provides Standardized Key Management API
    - Abstracts KMS details from the protocol consumer
  - Popularization: it takes two to tango
    - Key management tasks are often solved with in-house solutions that do not scale well
    - Increased awareness is a precondition of increased availability

# What's KMS?

- Protocol for managing keys that supports:
  - Key generation, retrieval, and persistent storage
  - Multiple key types (AES, RSA, SHA) and formats (ASCII, UTF-8)
  - Client-based key handling
  - Import of external keys
  - Attaching attributes and client data to a key
  - Init on-demand (start only if needed to optimize boot performance)

```
typedef struct _EFI_KMS_SERVICE_PROTOCOL
{
  EFI_KMS_GET_SERVICE_STATUS      GetServiceStatus;
  EFI_KMS_REGISTER_CLIENT         RegisterClient;
  EFI_KMS_CREATE_KEY              CreateKey;
  EFI_KMS_GET_KEY                 GetKey;
  EFI_KMS_ADD_KEY                 AddKey;
  EFI_KMS_DELETE_KEY              DeleteKey;
  EFI_KMS_GET_KEY_ATTRIBUTES      GetKeyAttributes;
  EFI_KMS_ADD_KEY_ATTRIBUTES      AddKeyAttributes;
  EFI_KMS_DELETE_KEY_ATTRIBUTES   DeleteKeyAttributes;
  EFI_KMS_GET_KEY_BY_ATTRIBUTES   GetKeyByAttributes;
  UINT32                          ProtocolVersion;
  EFI_GUID                        ServiceId;
  CHAR16                          *ServiceName;
  UINT32                          ServiceVersion;
  BOOLEAN                         ServiceAvailable;
  BOOLEAN                         ClientIdSupported;
  BOOLEAN                         ClientIdRequired;
  UINT16                          ClientIdMaxSize;
  UINT8                           ClientNameStringTypes;
  BOOLEAN                         ClientNameRequired;
  UINT16                          ClientNameMaxCount;
  BOOLEAN                         ClientDataSupported;
  UINTN                           ClientDataMaxSize;
  BOOLEAN                         KeyIdVariableLenSupported;
  UINTN                           KeyIdMaxSize;
  UINTN                           KeyFormatsCount;
  EFI_GUID                        *KeyFormats;
  BOOLEAN                         KeyAttributesSupported;
  UINT8                           KeyAttributeIdStringTypes;
  UINT16                          KeyAttributeIdMaxCount;
  UINTN                           KeyAttributesCount;
  EFI_KMS_KEY_ATTRIBUTE           *KeyAttributes;
} EFI_KMS_PROTOCOL;
```

# What Do You Do With KMS?

- Storage unlocking/decryption (Opal drives, password protected drives, secure RAID controllers)
- Management of machine-maintained passwords
- Management of user passwords
  - Secure Storage (KMS can be used to securely store user passwords)
  - Encryption facilitation (Passwords can be encrypted using KMS provided keys and then stored in the NVRAM or other unsecure storage)
- Device coupling (Couple device or system with a local KMS. For example, couple blade server with a specific server rack)
- Device attestation (secure handling of the device fingerprints)
- Firmware code or data encryption (encryption of the configuration data or critical portions of code)
- RPMC-based storage (manage keys and counters with KMS)

# Is It Safe To Use KMS?

- Make it secure
  - The main challenge: KMS should be readily available to the good guys and protected from the bad guys
  - Ways to secure KMS
    - Temporal protection
      - Disable some or all KMS facilities at certain boot stage (typically on transitioning between trust boundaries)
    - Client verification
      - RegisterClient interface allows for limited authentication of the caller
- Make it reliable
  - KMS deployment strategy should encompass provisioning and recovery scenarios
    - What happens when KMS provider is not available (temporary or permanently)?
    - What's the process of repurposing, replacement or ownership transfer for a KMS protected platform or device?

# How Do You Make a KMS?

- Multiple implementation options
  - Local (build KMS on top of device that is part of the system)
    - TPM based
    - BMC based
    - DC-SCM based
    - Special hardware based (peripheral device connected via standard interface, such as PCI or USB, or a custom interface)
  - Remote (build KMS on top of transport layer talking to a remote service provider)
    - Over the network (for example, based on KMIP protocol)

# TPM-Based KMS Implementation

# Why TPM and Not Any Other HSM Device?

- TPM is an obvious choice because of industry support and its ubiquity
- No extra BOM cost to add another Crypto Device
- TPM can create and securely store keys
- TPM has a flexible policy infrastructure that can be used to control access to TPM KMS objects
- Can be used independently or be used for redundancy Support to KMIP Server

# Considerations on Using TPM as a KMS HSM

- Directed by the platform need and trust policies for the platform

- Should be informed by current industry specifications from TCG
  - TCG PFP Specifications
  - TCG Platform Certificate Profile
  - TPM 2.0 Keys for Device Identity and Attestation
  - DICE Protection Environment
  - SP 800-133 Recommendation for Cryptographic Key Generation

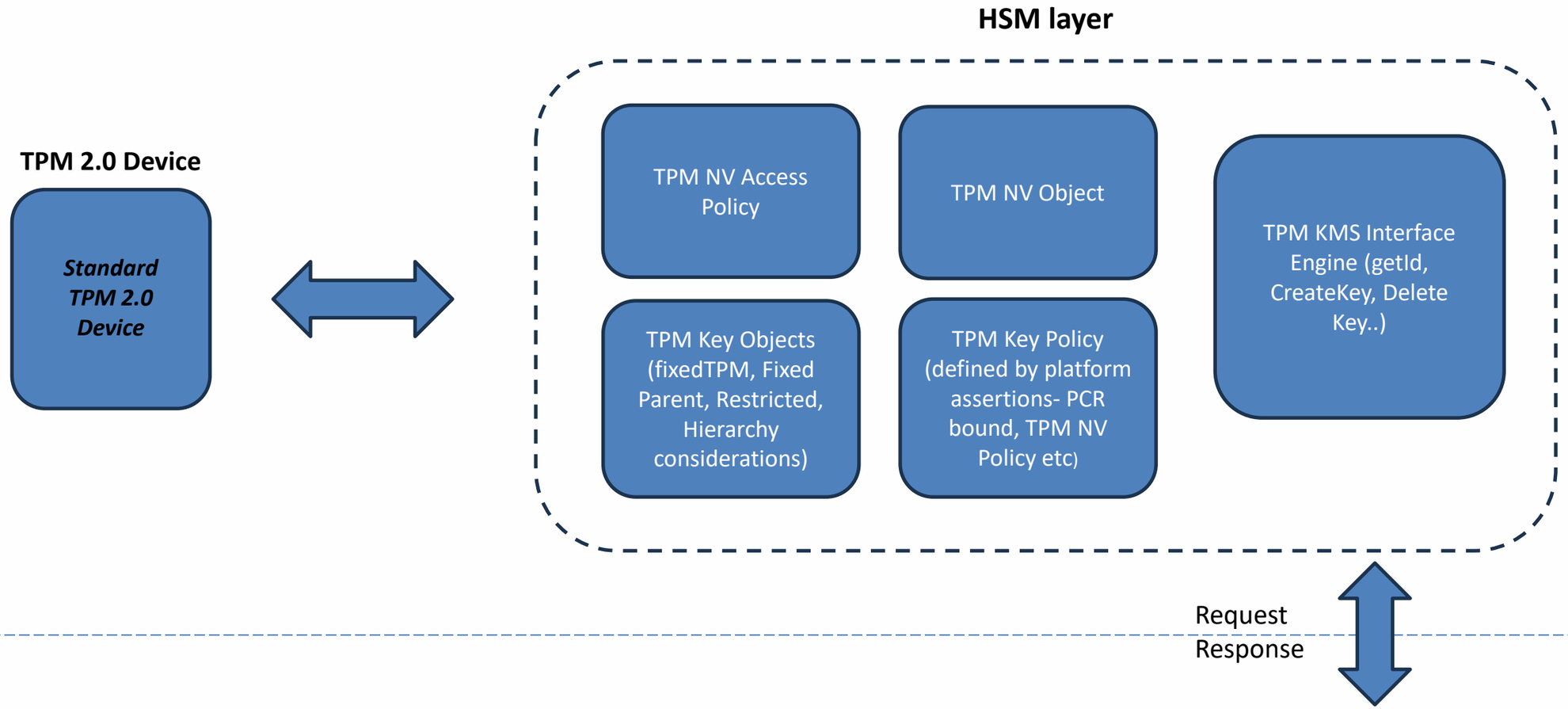
# Common Components That All Design Solutions Will Need

- TPM Policies
  - When should TPM objects for KMS be available?
  - Who should be able to use those objects and when should they be locked?

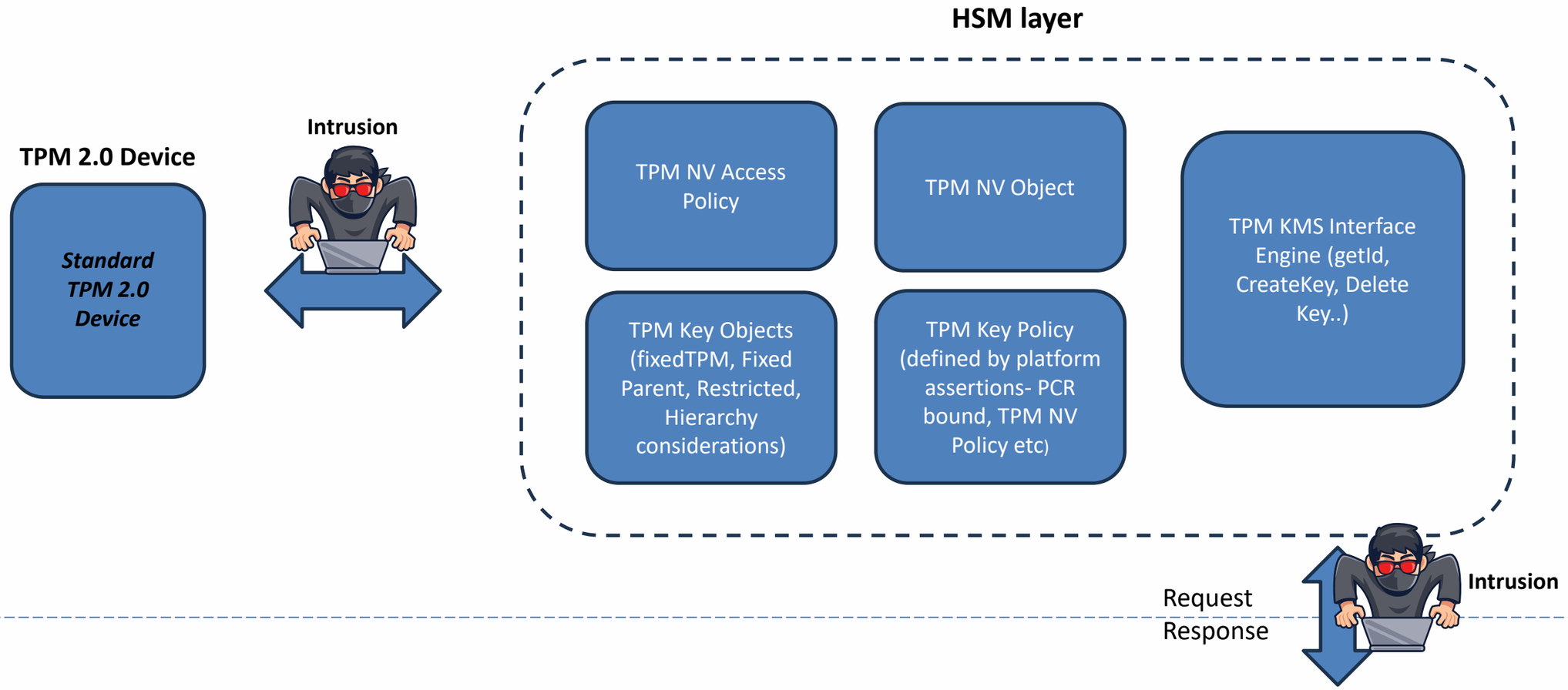
- Keys created in the TPM (Restrictions)
  - Primary key (Fixed, Restrictions, Policy)

- Hierarchy Consideration (Storage Hierarchy, Platform Hierarchy)

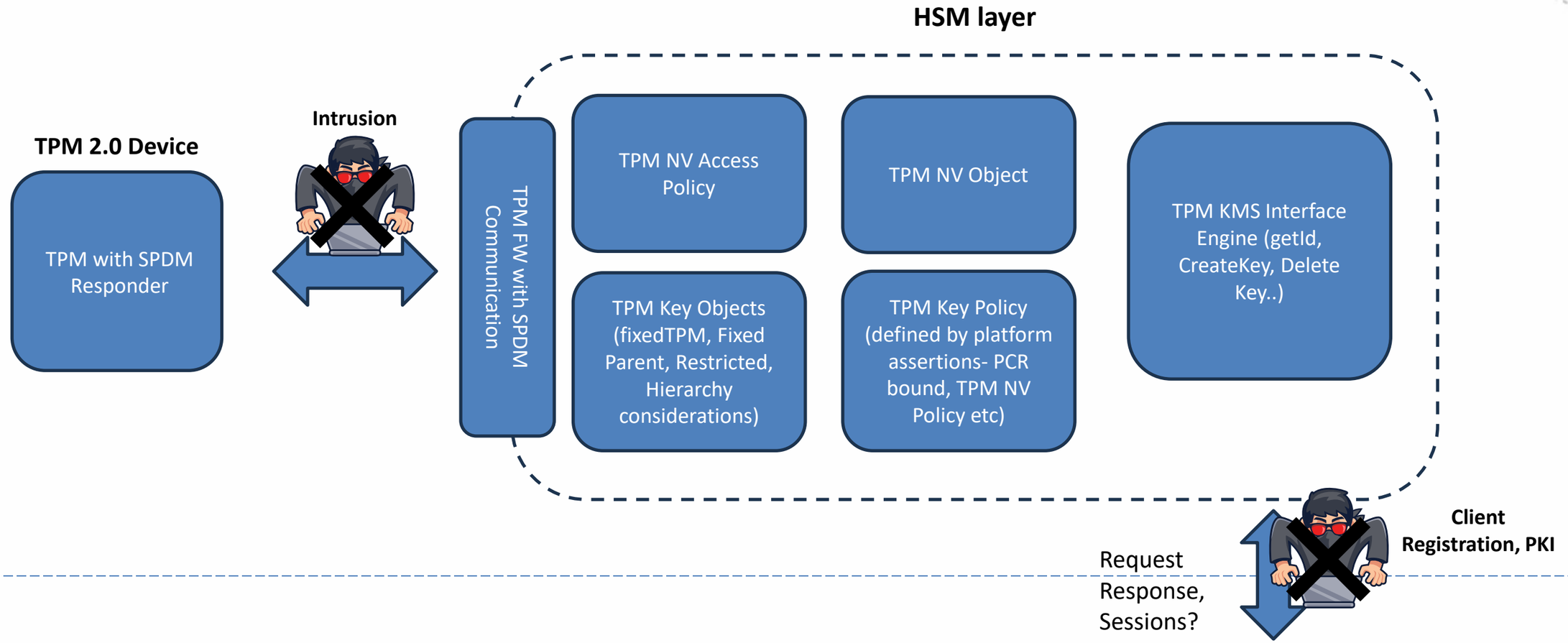
# Sample Simple Implementation



# Sample Implementation



# Making It Better



# Challenges

- Not one size fits all
  - Policies should be directed by platform security assertions and use case requirements
  - Considerations
    - Using PCR based policies can be brittle
    - TPM NV, TPM Authorize policies remove brittleness headache but also have their own Policy management headaches
      - (TPM Authorize, PolicySigned, allows an easier path to recovery for a broken TPM but that implies using another Key management infrastructure. Probably, KMIP)

# How To Make It Better?

- Improve availability
  - Today IHV's can't universally rely on KMS being part of the UEFI firmware
- Improve protocol description
  - Better document data persistence (what is persevered across boot boundaries), usage of client data and key attributes
  - Provide usage examples
- Build a threat model
- Consider potential interface improvements
  - Add support for longer keys and additional key types
  - Client-based locking (explicit interface to lock keys and/or services)
  - Programmatic mechanism to get more information about KMS provider (local/remove, underlying device, etc.)
    - One way to do it is by defining standard *ServiceId* GUIDs for a mainstream KMS types
  - Interface to get amount of a key storage available to a client
- Introduce MM version of the protocol in PI spec

Thanks for attending the UEFI Fall 2023 Developers Conference & Plugfest

For more information on UEFI Forum and UEFI Specifications, visit http://www.uefi.org

*presented by*