

[**https://docs.nvidia.com/jetson/archives/r34.1/DeveloperGuide/text/SD/Security/SecureBoot.html**](https://docs.nvidia.com/jetson/archives/r34.1/DeveloperGuide/text/SD/Security/SecureBoot.html)

<https://docs.nvidia.com/jetson/archives/r34.1/DeveloperGuide/text/SD/Security/OpTee.html#trusted-application-and-client-application-development>  
<https://docs.nvidia.com/jetson/archives/r34.1/DeveloperGuide/text/SD/Security/OpTee.html#key-derivation-function-of-a-fuse-key-and-user-defined-key>  
<https://docs.nvidia.com/jetson/archives/r34.1/DeveloperGuide/text/SD/Security/OpTee.html#secure-samples>  
<https://docs.nvidia.com/jetson/archives/r34.1/DeveloperGuide/text/SD/Security/OpTee.html#key-maintenance-and-ekbs>  
<https://docs.nvidia.com/jetson/archives/r34.1/DeveloperGuide/text/SD/Security/OpTee.html#aes-256-hardware-key-derivation-function-in-jetson-user-key-pta>  
<https://docs.nvidia.com/jetson/archives/r34.1/DeveloperGuide/text/SD/Security/DiskEncryption.html>

format the text properly and explain in detail

Here's a structured technical summary of OP-TEE on NVIDIA Jetson devices, formatted for clarity and technical documentation:

**OP-TEE on NVIDIA Jetson: Trusted Execution Environment Implementation**

**1. Architecture Overview**

**Two Execution Worlds:**

* **Normal World (NSE):** Runs standard OS (Linux)
* **Secure World (TEE):** Runs OP-TEE OS at ARMv8 Secure EL-1 level

**Key Components:**

* optee\_os: TEE OS handling secure operations (interrupts, crypto services)
* optee\_client: User-space components (libteec.so + tee-supplicant daemon)
* **Trusted Applications (TAs):** Run at Secure EL-0 level
* **Arm Trusted Firmware (ATF):** Handles secure/non-secure world transitions

**Communication Flow:**

1. Client App → TEE Client API → Linux Kernel Driver → ATF → OP-TEE OS → TA
2. Response follows reverse path

**2. Trusted Application Development**

**TA Types:**

|  |  |  |  |
| --- | --- | --- | --- |
| Type | Privilege Level | API Access | Use Case |
| User Mode TA | Secure EL-0 | GlobalPlatform API | General secure applications |
| Pseudo TA (PTA) | Secure EL-1 | Direct HW access | Low-level hardware control |

**Development Requirements:**

1. Conform to GlobalPlatform TEE Internal Core API
2. Use NIST-SP 800-108 compliant KDF for crypto operations
3. Implement UUID-based identification system

**Porting Existing TAs:**

* Replace IPC mechanisms with RPC
* Convert crypto implementations to use:

TEE\_GenerateRandom() // Hardware RNG access  
TEE\_AEInit() // Authenticated Encryption

**3. Encrypted Keyblob (EKB) System**

**Key Hierarchy:**

1. **KEK2 Fuse Key** (128-bit, hardware-burned)
2. **KEK2\_RK** = AES-128-ECB(FV, KEK2)
3. **EKB\_EK/EKB\_AK** = NIST-SP-800-108(KEK2\_RK, ...)

**EKB Structure:**

[16-byte Header]  
[16-byte AES-CMAC]  
[Random IV]  
[Encrypted Payload]

**Generation Process:**

python3 gen\_ekb.py -kek2\_key fuse.bin -fv fv.bin -in\_sym\_key key.bin -out eks.img

**Verification Commands:**

# Check EKB size constraints  
stat -c%s eks.img | awk '{if ($1 < 1024 || $1 > 1048576) exit 1}'  
  
# Validate CMAC  
openssl dgst -mac cmac -macopt cipher:aes-128-cbc -macopt hexkey:$EKB\_AK -binary < ekb\_content

**4. Secure Sample Implementations**

**1. jetson\_user\_key\_pta (Pseudo TA)**

**Key Management Functions:**

#define JETSON\_USER\_KEY\_CMD\_GET\_EKB\_KEY 0  
#define JETSON\_USER\_KEY\_CMD\_GEN\_KEY 3

**SE Keyslot Clearing:**

tegra\_se\_clear\_aes\_keyslots(SE\_KEYSLOT\_11 | SE\_KEYSLOT\_12);

**2. hwkey-agent (User TA)**

**Service Interface:**

#define HWKEY\_AGENT\_TA\_CMD\_ENCRYPTION 0  
#define HWKEY\_AGENT\_TA\_CMD\_DECRYPTION 1

**Usage Example:**

nvhwkey-app -e -i plain.txt -o cipher.bin -k EKB\_USER\_KEY\_DISK\_ENCRYPTION

**5. Hardware-Accelerated Cryptography**

**Security Engine (SE) Integration:**

TEE\_Result tegra\_se\_aes\_ecb\_kdf(  
 uint8\_t \*derived\_key,   
 size\_t key\_len,  
 uint8\_t \*iv,  
 size\_t iv\_len,  
 uint32\_t keyslot  
);

**NIST-SP 800-108 KDF Implementation:**

TEE\_Result nist\_sp\_800\_108\_cmac\_kdf(  
 uint8\_t \*key\_in,  
 uint32\_t key\_len,  
 char const \*context,  
 char const \*label,  
 uint32\_t dk\_len,  
 uint8\_t \*key\_out  
);

**Security Best Practices**

1. **Keyslot Management:**
   * Clear keyslots immediately after use
   * Use tegra\_se\_clear\_aes\_keyslots() post-operation
2. **EKB Handling:**
   * Minimum 1024-byte size with random padding
   * Store FVs in secure persistent storage
3. **Runtime Security:**

# Disable SE clocks post-boot  
echo 0 > /sys/class/tegra-se/clk\_enable

**Note:** Secure Boot and Disk Encryption features mentioned in documentation are currently unsupported in Jetson Linux R34.1. Refer to NVIDIA's security advisories for updates.

This structure provides comprehensive technical details while maintaining readability. For implementation, refer to the [Jetson Linux Security Guide](https://developer.nvidia.com/embedded/jetson-linux-security) and [OP-TEE Official Documentation](https://optee.readthedocs.io).

⁂