**NVIDIA Jetson Security: OP-TEE, Secure Boot, and Disk Encryption**

**1. OP-TEE Trusted Execution Environment (TEE) Implementation**

**Architecture Overview**

* **Two Execution Worlds:**
  + **Normal World (NSE):** Runs standard Linux OS.
  + **Secure World (TEE):** Runs OP-TEE OS at ARMv8 Secure EL-1.
* **Key Components:**
  + optee\_os: Secure OS handling interrupts and cryptographic services.
  + optee\_client: User-space components (libteec.so and tee-supplicant daemon).
  + **Trusted Applications (TAs):** Run at Secure EL-0.
  + **Arm Trusted Firmware (ATF):** Manages transitions between secure and non-secure worlds.
* **Communication Flow:**
  + Client App → TEE Client API → Linux Kernel Driver → ATF → OP-TEE OS → TA
  + Responses follow the reverse path.

**2. Trusted Application (TA) Development**

**TA Types**

* **User Mode TA:** Runs at Secure EL-0, uses GlobalPlatform APIs, for general secure applications.
* **Pseudo TA (PTA):** Runs at Secure EL-1, has direct hardware access, for low-level control.

**Development Requirements**

* Conform to GlobalPlatform TEE Internal Core API.
* Use NIST-SP 800-108 compliant Key Derivation Function (KDF) for cryptographic operations.
* Implement UUID-based identification for each TA.

**Porting Existing TAs**

* Replace IPC mechanisms with Remote Procedure Calls (RPC).
* Update cryptographic code to use:

c

TEE\_GenerateRandom() *// Hardware RNG access*

TEE\_AEInit() *// Authenticated Encryption*

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

**Key Hierarchy**

* **KEK2 Fuse Key:** 128-bit hardware-burned key.
* **KEK2\_RK:** Derived as AES-128-ECB(FV, KEK2).
* **EKB\_EK/EKB\_AK:** Derived using NIST-SP-800-108(KEK2\_RK, ...).

**EKB Structure**

* 16-byte Header
* 16-byte AES-CMAC
* Random Initialization Vector (IV)
* Encrypted Payload

**Generation Process**

* Generate EKB image:

bash

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:

bash

stat -c%s eks.img | awk '{if ($1 < 1024 || $1 > 1048576) exit 1}'

* Validate CMAC:

bash

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:**

c

#**define** JETSON\_USER\_KEY\_CMD\_GET\_EKB\_KEY 0

#**define** JETSON\_USER\_KEY\_CMD\_GEN\_KEY 3

* **SE Keyslot Clearing:**

c

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

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

* **Service Interface:**

c

#**define** HWKEY\_AGENT\_TA\_CMD\_ENCRYPTION 0

#**define** HWKEY\_AGENT\_TA\_CMD\_DECRYPTION 1

* **Usage Example:**

bash

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

**5. Hardware-Accelerated Cryptography**

**Security Engine (SE) Integration**

* **Key Derivation Function Example:**

c

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:**

c

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

);

**6. Security Best Practices**

* **Keyslot Management:**
  + Clear keyslots immediately after use.
  + Use tegra\_se\_clear\_aes\_keyslots() post-operation.
* **EKB Handling:**
  + Minimum EKB size is 1024 bytes, with random padding for security.
  + Store Firmware Variables (FVs) in secure persistent storage.
* **Runtime Security:**
  + Disable Security Engine clocks post-boot:

bash

echo 0 > /sys/class/tegra-se/clk\_enable

**7. Notes & Limitations**

* **Secure Boot and Disk Encryption** features mentioned in documentation are currently unsupported in Jetson Linux R34.1.
* Always refer to NVIDIA security advisories for updates.

**This structure provides comprehensive technical details, clear bullet points, and command blocks for easy chunking and chatbot extraction.**