LVC20-204
Encrypted firmwares and how to bake them right

Sumit Garg
Linaro Ltd.

Linaro Connect
VIRTUAL 2020

Overview

- Security basics
  - Terms
- Firmware encryption
  - Assets
  - Use-cases
- Challenges
  - Secret key protection?
  - Device unique or class wide key?
  - Play nicely with firmware signature?
  - Firmware updates?
- Implementation
  - TF-A
  - OP-TEE

Security basics

- Security is **not** a turn key solution but rather made of many different components
- There is no such thing as "**a secure system**", only **secure enough**
- Threat modelling is essential
  - Knowledge of system **assets**
  - **Threats** to them
  - **Mitigated** via security features
- Security features (such as firmware encryption)
  - A **tool** in your toolbox

Terms

Classic security concepts:

- Confidentiality
- Integrity
- Authentication
- Authorization
- Non-repudiation

Firmware encryption

Classic security concepts:

- Confidentiality
- Integrity
- Authentication
- Authorization
- Non-repudiation

Firmware encryption allows us to achieve these properties for a firmware, using:

- **Symmetric** encryption
  - Reason to **not** use **asymmetric** encryption: boot time limitation.
- **Authenticated Encryption (eg. AES-GCM)**
  - Ensures integrity of encrypted firmware blob.

Firmware encryption

Assets?

Possible firmware assets to protect:

- **Software IP**
  - Allow confidentiality protection for software IP.
- **Device secrets**
  - Allow firmware image to act as secret store (though unlikely to be suitable for high value secrets).
- **Implementation details**
  - Make it harder to develop exploits for any vulnerabilities in the firmware.

Firmware encryption

Use-cases?

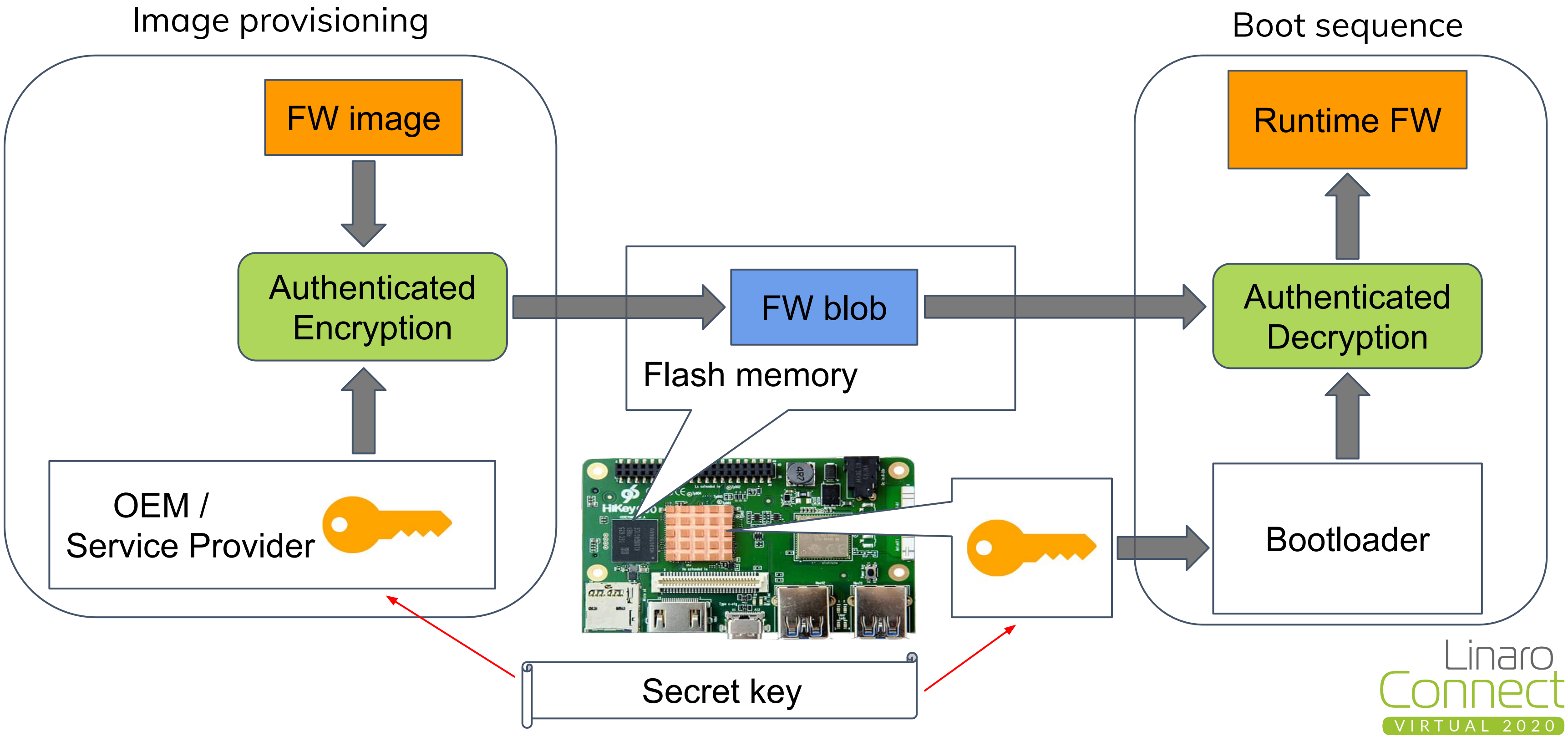
The major drivers for this feature are the emerging robustness requirements for software **Digital Rights Management** (DRM) implementations.

Make it even harder to reverse engineer Trusted Execution Environment (TEE) and therefore would like to see that **Trusted OS** and corresponding **Trusted Applications** are not just signed, but also **encrypted**.

TEE assets:

- DRM software IP.
- DRM implementation details.

Firmware encryption



Challenge: Secret key protection?

Secret key protection may vary from one platform to another depending on use-case and hardware capabilities like:

- Key is derived from **device secrets** like OTP or such.
- Key is provisioned into **secure fuses** on the device.
- Key is provisioned into **hardware crypto accelerator**.
- Key is provisioned into platform **secure storage** like non-volatile SRAM etc.

In order to address this varying requirement, we need to provide an **abstraction layer** to retrieve **secret key / secret key handle** and platform can provide underlying implementation.

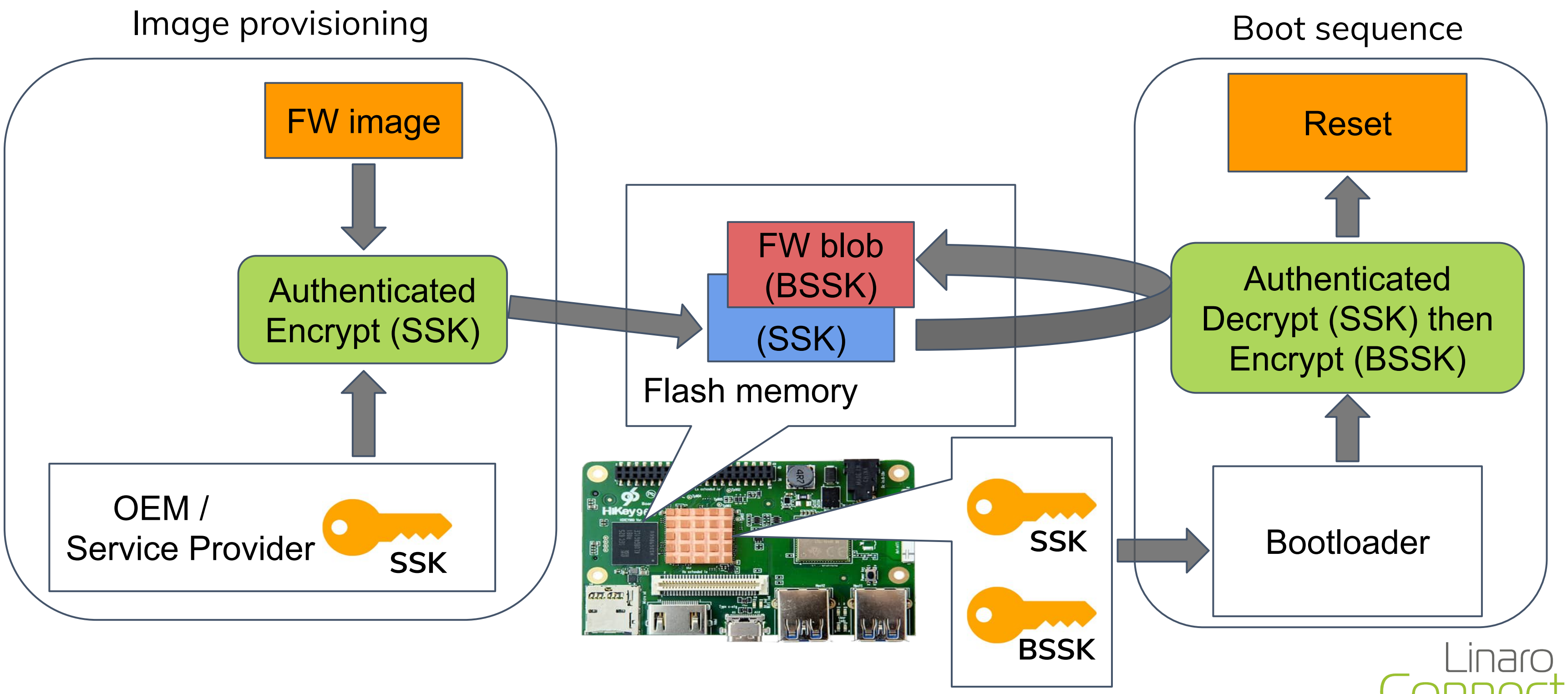
Challenge: Device unique or class wide key?

Secret key type?

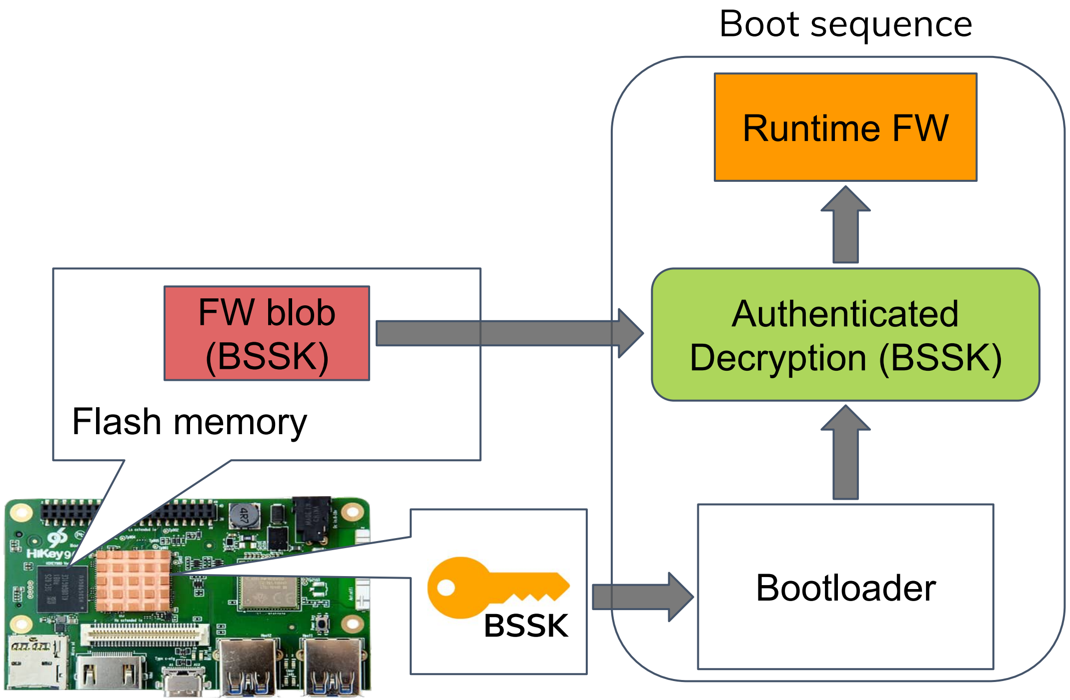
- **Device unique key:** Unique per device, aka Binding Secret Symmetric Key (**BSSK**)
  - **Pros**: limits attacks surface to per device, provides protection against software cloning.
  - **Cons**: scalability issue to manage per device unique firmware images.
- **Class wide key:** Common shared key for a class of devices, aka Shared Secret Key (**SSK**)
  - **Pros**: single firmware image, easy to deploy and update.
  - **Cons**: comparatively larger attack surface, class wide attacks.

How about leveraging benefits of both key types?

Firmware encryption: first boot (firmware binding)



Firmware encryption: subsequent boot



Firmware encryption + signature

Classic security concepts:

- Confidentiality
- Integrity
- Authentication
- Authorization
- Non-repudiation

Firmware **encryption** ensures these properties on behalf of OEM / Service Provider.

- Maintain firmware **confidentiality**.

Firmware **signature** ensures these properties on behalf of OEM / Service Provider.

- Allow only **authorized** firmware to execute.

Challenge: encryption + signature?

Encryption and **signature** schemes are well known cryptographic constructs but when their combination is to be used:

- Proper attention is required towards **achievable** security properties

Possible combinations:

- **Encrypt-then-sign**
- **Sign-then-encrypt**
- **Sign-then-encrypt-then-MAC**

Challenge: encryption + signature?

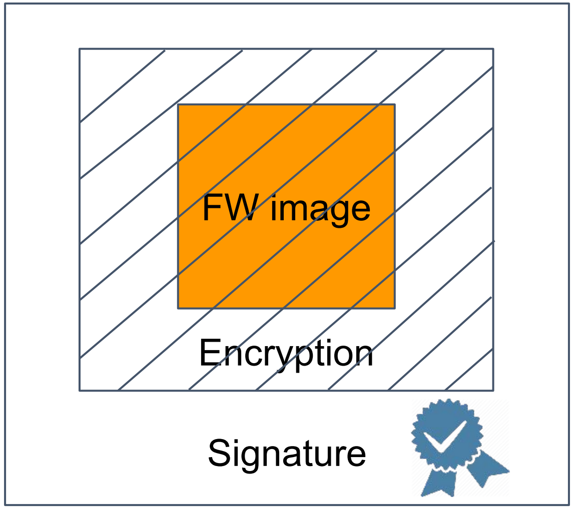
Encrypt-then-sign

Security properties:

- Confidentiality
- Integrity
- Authentication
- Authorization

Shortcomings:

- Only encrypted firmware blob is **non-repudiable** to OEM / SP.
- Signing encrypted blob makes it **immutable**
  - Doesn't allow **re-encryption** on device, aka firmware binding.



Challenge: encryption + signature?

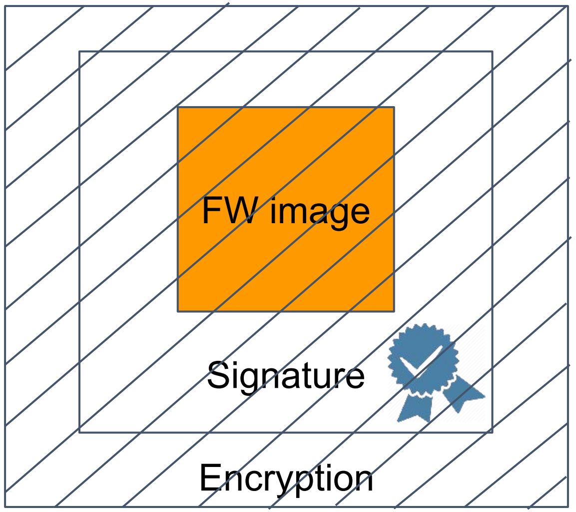
Sign-then-encrypt

Security properties:

- Confidentiality
- Authentication
- Authorization
- Non-repudiation

Shortcomings:

- **Plain** encryption doesn't assure integrity of encrypted blob.
  - Vulnerable to **Chosen Ciphertext Attacks** (CCAs).



Challenge: encryption + signature?

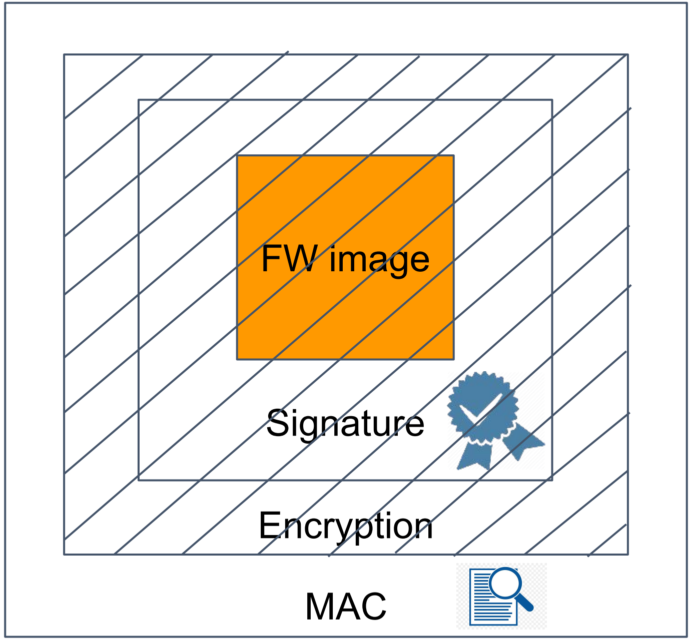
Sign-then-encrypt-then-MAC

Security properties:

- Confidentiality
- Integrity
- Authentication
- Authorization
- Non-repudiation

Concerns addressed:

- MAC tag assures **integrity** of encrypted blob.
- Allows firmware **re-encryption**.



Challenge: Firmware updates?

Generally, following approaches are used to apply firmware updates:

- Update complete firmware **partition**
  - Firmware encryption **doesn’t** adds any complexity
    - Updater could verify overall firmware partition signature.
- Update **individual** firmware images
  - Firmware encryption **adds** complexity:
    - Updater needs to verify each individual image, requires access to encryption key.
    - Either updater needs to be a secure world entity or leverages secure world decrypt and verify service.

Implementation: TF-A

Trusted Firmware-A (TF-A) supports an **I/O encryption layer** (drivers/io/io_encrypted.c):

- Layered on top of any base I/O layer (eg. drivers/io/io_fip.c)
  - To allow loading of corresponding encrypted firmware payload.
- Approach used: **sign-then-encrypt-then-MAC**.
- Uses **encrypt_fw** tool (tools/encrypt_fw/) to encrypt firmwares during build.
- Build options:
  - **DECRYPTION_SUPPORT**: enables firmware decryption layer (values: **aes_gcm** or **none**)
  - **FW_ENC_STATUS**: firmware encryption key flag (values: 0 -> **SSK**, 1 -> **BSSK**)
  - **ENC_KEY**: 32-byte (256-bit) symmetric key
  - **ENC_NONCE**: 12-byte (96-bit) encryption nonce or Initialization Vector (IV)
  - **ENCRYPT_{BL31/BL32}**: flag to enable BL31/BL32 encryption

Implementation: OP-TEE

OP-TEE supports REE-FS Trusted Application (TA) image type: **SHDR_ENCRYPTED_TA**

- Allows loading of encrypted REE-FS TAs.
- Approach used: **sign-then-encrypt-then-MAC**.
- Uses **scripts/sign_encrypt.py** to encrypt TAs during build.
    - Generates random nonce for every encryption.
- TA build options:
    - **CFG_ENCRYPT_TA**: flag to enable encryption of corresponding TA.
    - **TA_ENC_KEY**: 32-byte (256-bit) symmetric key.

Future work…

- Let’s champion open source security frameworks
    - Reduces efforts to maintain custom solutions
- Encryption framework: contributions are welcome, adding:
    - Framework improvement
    - Platform support

Thank you

Accelerating deployment in the Arm Ecosystem

support@linaro.org

Linaro Connect VIRTUAL 2020