Migration to PSA Crypto API

Manish Badarkhe

19/10/2023

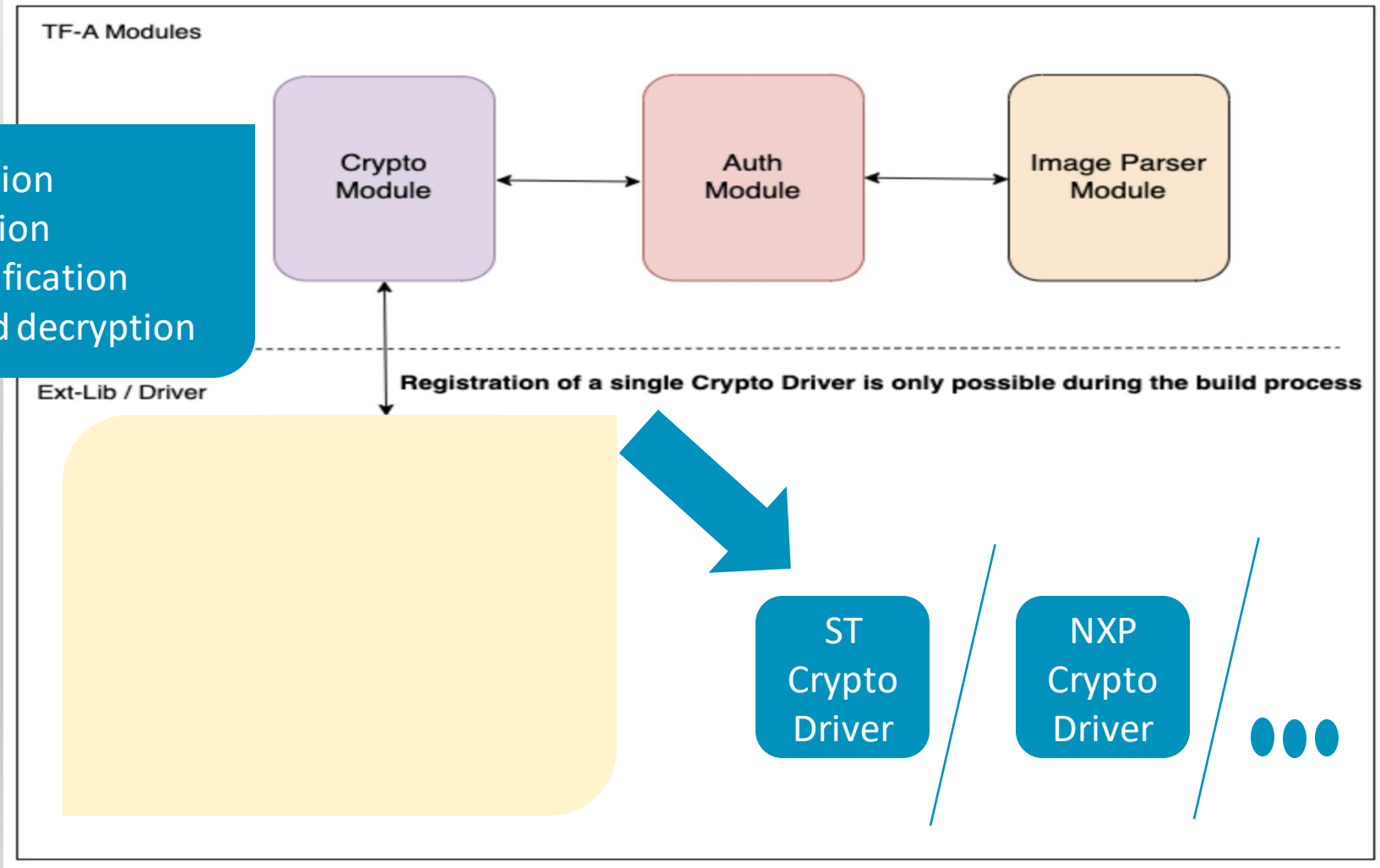
Agenda

- Existing Crypto Mechanism - using legacy Crypto API
- What is PSA?
- Design of PSA Crypto API
- Use of PSA Crypto API in TF-A
- PSA Crypto API verification using CI
- Future Scope: MbedTLS with PSA Crypto driver

Existing Crypto Mechanism

- Register a Crypto Module for leveraging Crypto API function & SW driver in mbedTLS lib
- The default TF-A crypto driver can be substituted by a platform-specific one
- Multiple Crypto Modules registration are not possible

1. Hash Verification
2. Hash Calculation
3. Signature Verification
4. Authenticated decryption



Challenges and Solution

Challenges

- Support dispatching crypto operations to different crypto hardware IPs
- Support 2 key storage options:
  1. locally stored on the application processor.
  2. externally stored inside a protected environment such as a secure element.

  Current implementation focuses on option (1) so far
- Hardware and software backends can coexist in the same firmware, accessible beneath a unified API

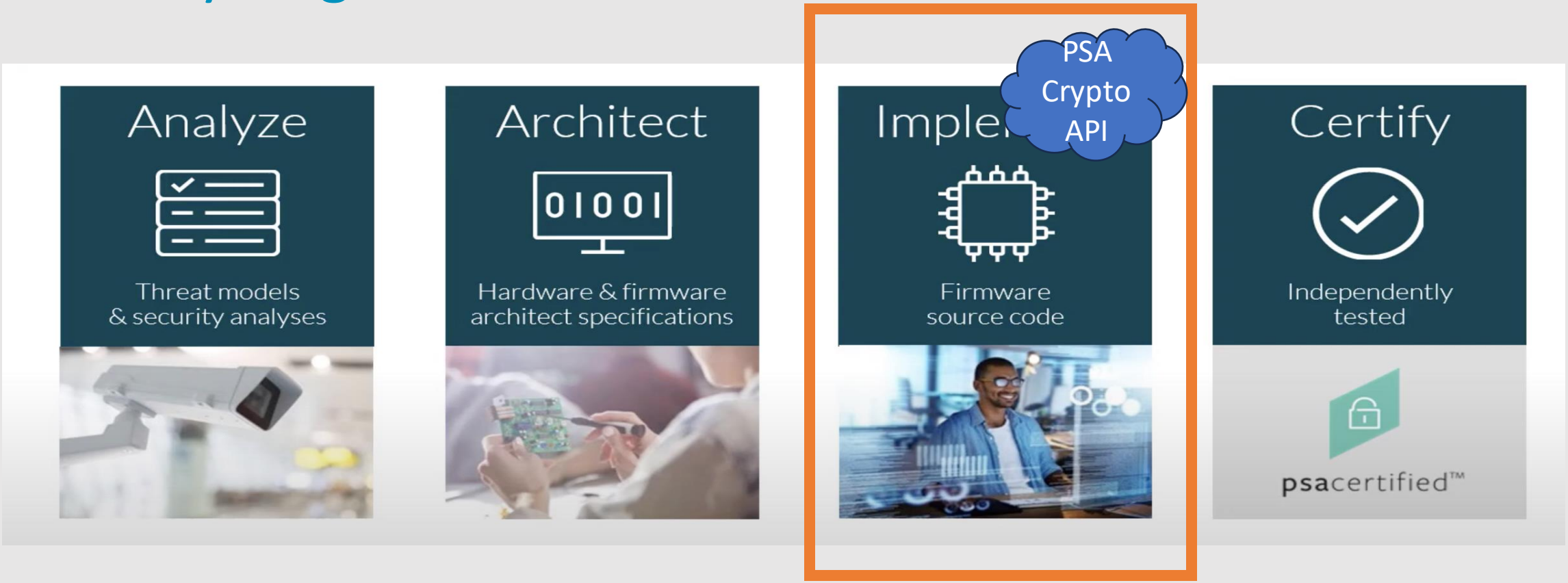
Solution

- Use PSA Crypto API implementation

What is PSA ?

- PSA Certified is an independent evaluation and certification scheme developed by Arm and its security partners
- PSA provides a recipe, based on industry best practice.
- Allows security to be consistently designed in, at both a hardware and firmware level.
- It is cost effective

Four Key Stages



- A holistic set of threat models, security analyses, hardware and firmware architecture specifications, an open-source firmware reference implementation, and an independent evaluation and certification scheme

PSA Crypto API Implementation

- Mbed TLS PSA Cryptography API implementation is made of
  - **Core**
  - **PSA drivers**

- Core -
  - It is responsible for ensuring sanity of the arguments and pass them properly to the appropriate PSA drivers
  - Building the arguments for the call to PSA driver interface
  - Does not perform any cryptographic operation on its own

- PSA Crypto Driver -
  - Responsible for the Cryptographic operations

PSA Crypto API Template

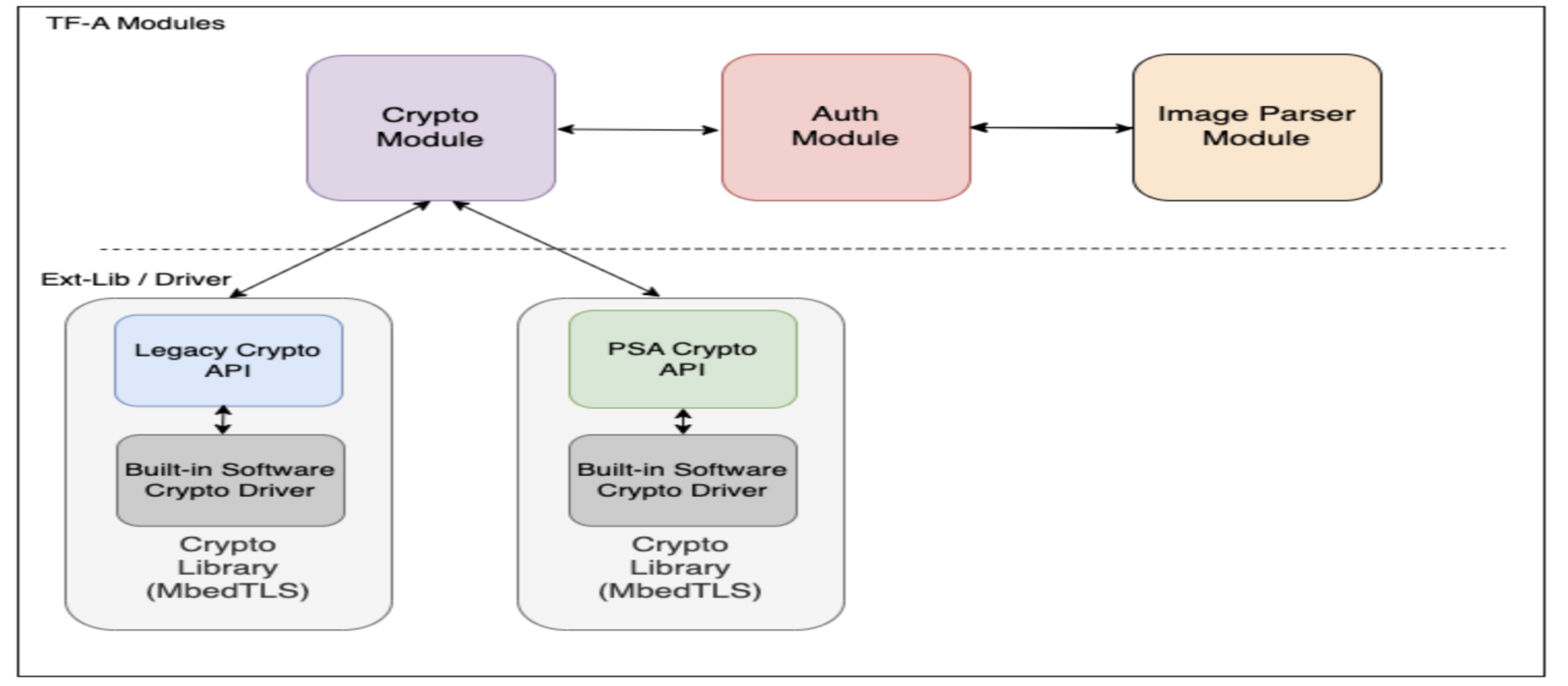
```
psa_status_t psa_api( ... )
{
    psa_status_t status;

    /* Pre driver interface call processing: validation of arguments, building
     * of arguments for the call to the driver interface, ... */
    ...
    /* Call to the driver interface */
    status = psa_driver_wrapper_<entry_point>( ... );
    if( status != PSA_SUCCESS )
        return( status );

    /* Post driver interface call processing: validation of the values returned
     * by the driver, finalization of the values to return to the caller,
     * clean-up in case of error ... */
}
```

Integration of PSA Crypto API into TF-A

- Create a new alternate Crypto Module for leveraging PSA Crypto API function & SW driver in mbedTLS lib



arm

Use of PSA Crypto API

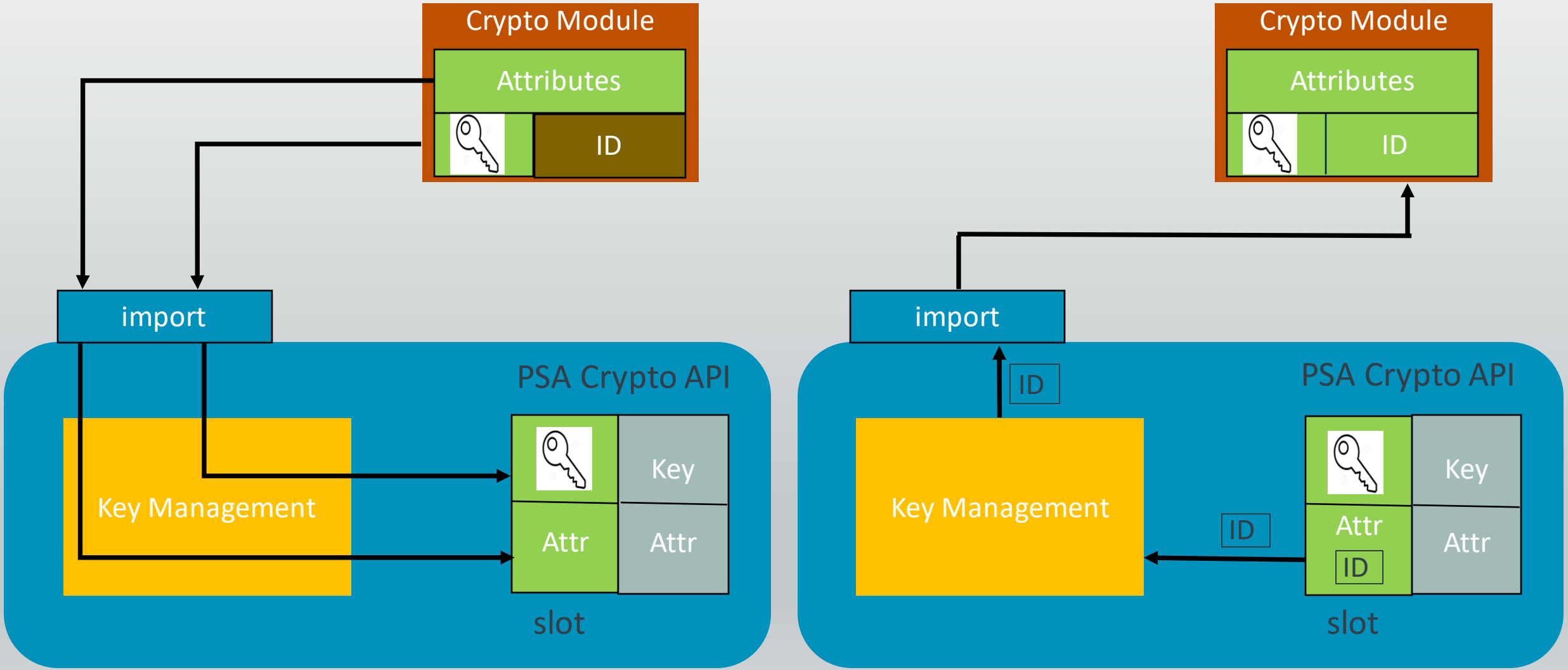
- Signature Verification –
  - psa_set_key_algorithm()
  - psa_set_key_type()
  - psa_set_key_usage_flags()
  - psa_import_key()
  - psa_verify_message()
- Hash Computation –
  - psa_hash_compute()
- Hash Verification –
  - psa_hash_compare()

Key Attributes

- Key Attributes
  - Key Type
  - Key Size
  - Key Lifetime – Persistent and its location
  - Key Policy
  - Key Algorithm

- Example Key Attributes with RSA Public Key verification -
  - Type: PSA_KEY_TYPE_RSA_PUBLIC_KEY
  - Lifetime: PSA_KEY_PERSISTENCE_VOLATILE with location (0x0)
  - Policy: PSA_KEY_USAGE_VERIFY_MESSAGE
  - Algorithm: PSA_ALG_RSA_PSS(hash_alg)

Use Public Key using Key-ID - Transparent Driver



Test Configs

- Measured-Boot + PSA Crypto
  - tf-l1-boot-tests-misc/fvp-psa-mbedtls-mb_hash384-optee:fvp-optee.mb-linux.rootfs+ftpm_384-fip.ftpm-aemv8a
- Trusted Board Boot + PSA Crypto (RSA)
  - tf-l3-boot-tests-misc/fvp-tbb-psa-mbedtls,fvp-default:fvp-tftf-fip.tftf-aemv8a-debug
- Trusted Board Boot + PSA Crypto (ECDSA)
  - tf-l3-boot-tests-misc/fvp-tbb-psa-mbedtls-ecdsa,fvp-default:fvp-tftf-fip.tftf-aemv8a-debug
- Trusted Board Boot + PSA Crypto [RSA ROT PK + ECDSA Certs]
  - tf-l3-boot-tests-misc/fvp-tbb-psa-mbedtls-rsa-ecdsa-with-rsa-rotpk-ecdsa-cert,fvp-default:fvp-tftf-fip.tftf-aemv8a-debug

References

- Changes posted for review -
  - https://review.trustedfirmware.org/q/topic:%22mb/psa-crypto-ecdsa%22
  - https://review.trustedfirmware.org/q/topic:%22mb/psa-crypto-support%22

- PSA Crypto API references –
  - https://github.com/Mbed-TLS/mbedtls/releases/tag/v3.4.1
  - https://armmbed.github.io/mbed-crypto/html/

Next Steps

- Test the whole stack on v3.5.0 mbedTLS release
- Remove temporary helper functions in the TF-A Crypto Driver and instead use the mbedTLS PSA API function wherever applicable
- Plans to improve Key management support
- Use PSA Crypto API for Authenticated Decryption support

Template PSA driver wrapper - transparent driver

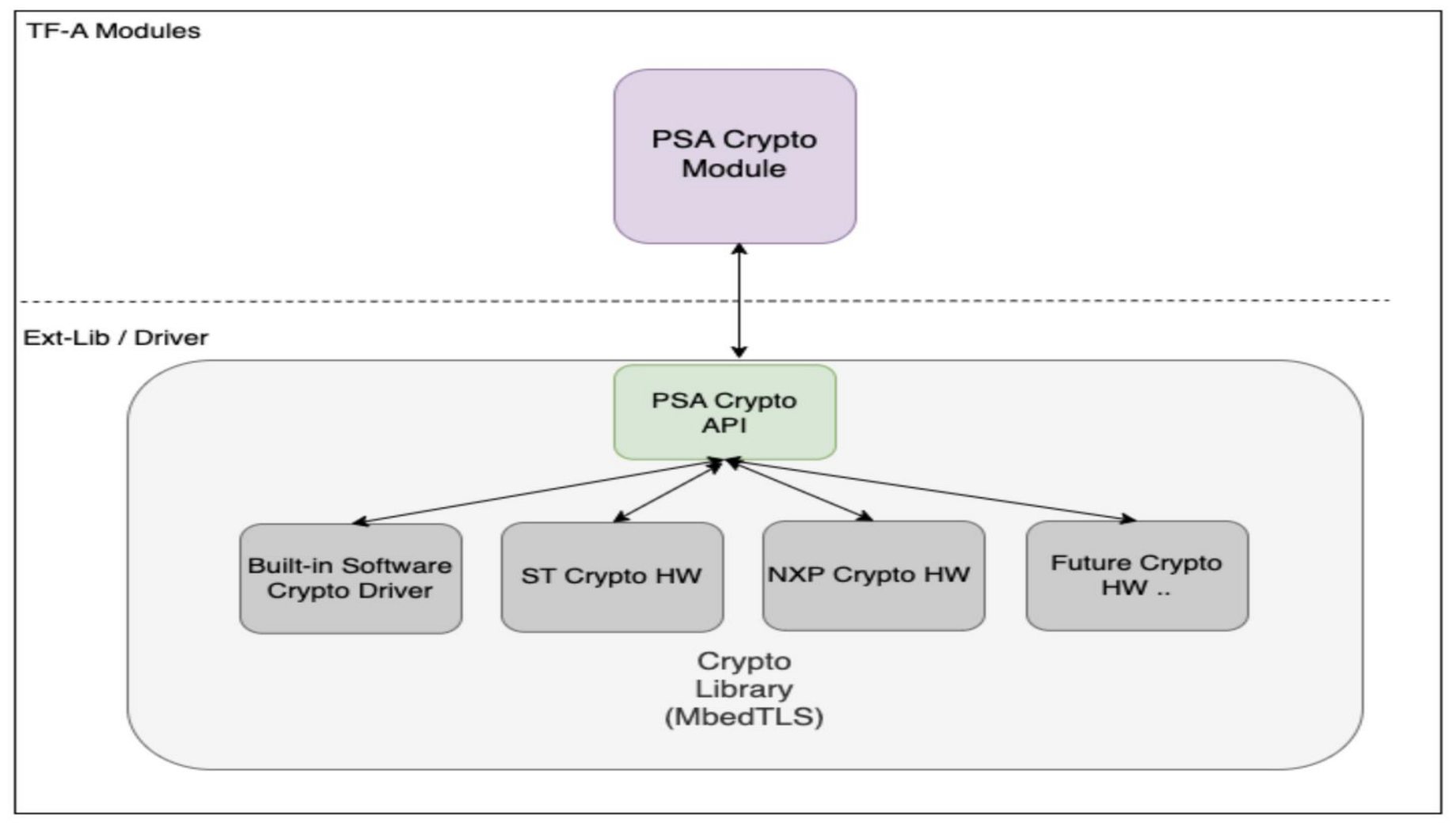
```
psa_driver_wrapper_xxx() -

switch(location)
case PSA_KEY_LOCATION_LOCAL_STORAGE                    //transparent driver

  #if defined(PSA_CRYPTO_ACCELERATOR_DRIVER_PRESENT)   // HW - Driver
    #if defined(X_DRIVER_PREFIX_ENABLED)
      if(/* conditions for X driver capabilities */)
        X_driver_transparent_xxx()                     //call to driver entry point
      if (status != PSA_ERROR_NOT_SUPPORTED) return status
    #endif

    #if defined(Y_DRIVER_PREFIX_ENABLED)
      if((/* conditions for Y driver capabilities */))
        Y_driver_transparent_xxx()                     //call to driver entry point
      if (status != PSA_ERROR_NOT_SUPPORTED) return status
    #endif
  #endif
```

Possible future Look



Thank You
Danke
Gracias
Grazie
谢谢
ありがとう
Asante
Merci
감사합니다
धन्यवाद
Kiitos
شكرًا
ধন্যবাদ
תודה

Template PSA driver wrapper – Opaque driver

```
case SECURE_ELEMENT_LOCATION                          //opaque driver

#if defined(PSA_CRYPTO_ACCELERATOR_DRIVER_PRESENT)
    #if defined(Z_DRIVER_PREFIX_ENABLED)
        if(/* conditions for Z driver capabilities */)
            Z_driver_transparent_xxx()                //call to driver entry point
        if (status != PSA_ERROR_NOT_SUPPORTED) return status
    #endif
#endif

return psa_xxx_builtin()                              // fall back to built in implementation
```