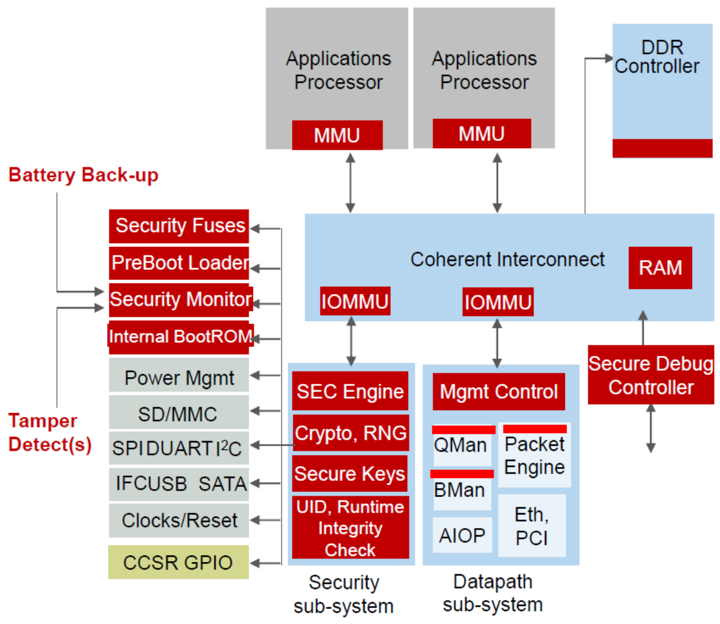
Secure Edge Computing Solution on LayerScape Platforms

Secure edge computing solution is required in secure manufacturing, secure Enrollment, secure device monitoring, secure container and application deployment.
This documents introduces trust architecture on Layerscape platform, trust software solution and user application development with OpenSSL Engine to offload encrypt/decrypt on hardware secure module.

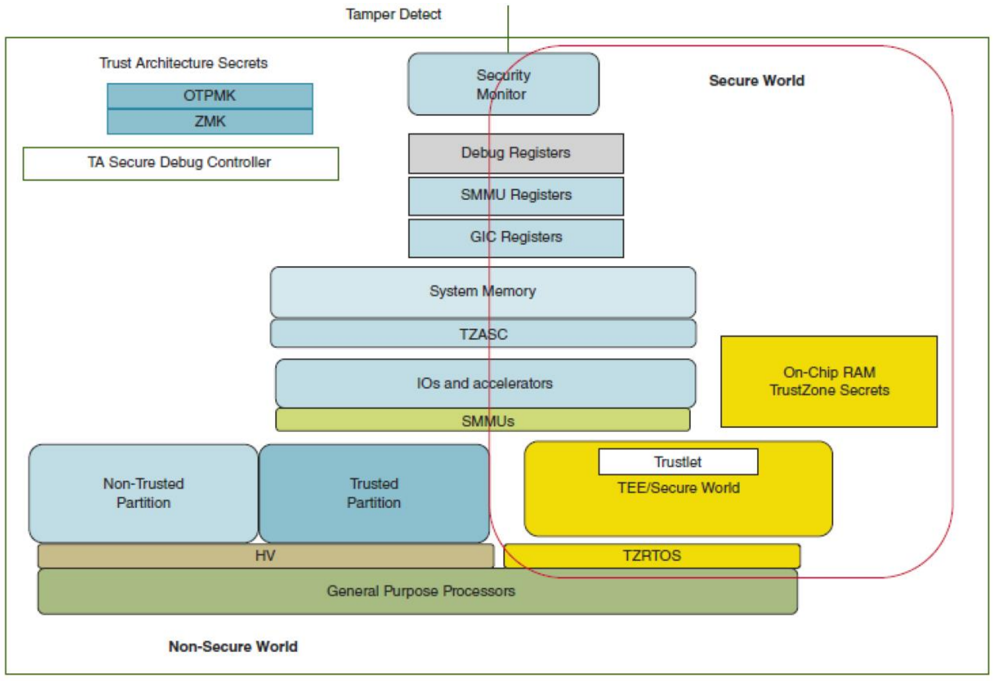
1. Layerscape Trust Architecture

Layerscape hardware trust architecture simplifies secure products development, the following figure describes the Layerscape hardware architecture. All Layerscape SoCs support trust architecture to provide secure boot, secure storage, key protection, key revocation, secure debug, tamper detection, strong partitioning and manufacturing protection.



In Arm TrustZone software and hardware are isolated as secure and non-trusted two worlds.
TrustZone Address Space Controller is divided into trusted and non-trusted two parts.
Trust secure world is a security domain within the overall Trust Architecure SoC.
Trust Architecture provides hardware secure boot, hardware debug protection, external tamper detection, and device secrets which even TZ Secure World software isn't able to access.
The Security monitor is a companion module to the security engine(SEC) module. The secure monitor is the SOC's central reporting point for security-relevant events such as the success or failure of boot software validation and the detection of potential security compromises. SEC can operate in the Trusted, Secure, Non-Secure and Fail modes.
Master key is an AES-256 key provisioned by OEMs. Master key is only used by the SEC

to encrypt and decrypt blob keys. It is never used directly on user selected data.



Master key can be One Time Programmable Master Key(OTPMK), Zeroizable Master Key(ZMK) and Combined Master Key(CMK).

SEC can protect data in a cryptographic data structure called a blob, which provides both confidentiality and integrity protection. The data to be protected is encrypted so that it can be safely placed into non-volatile storage before the chip is powered down. Blobs are cryptographically separated per security domain.

Black Key mechanism is intended for protection of user keys against bus snooping while keys are being written to or read from memory external to the SOC.

Keys stored in memory is encrypted form and decrypted on-the-fly when used by CAAM.

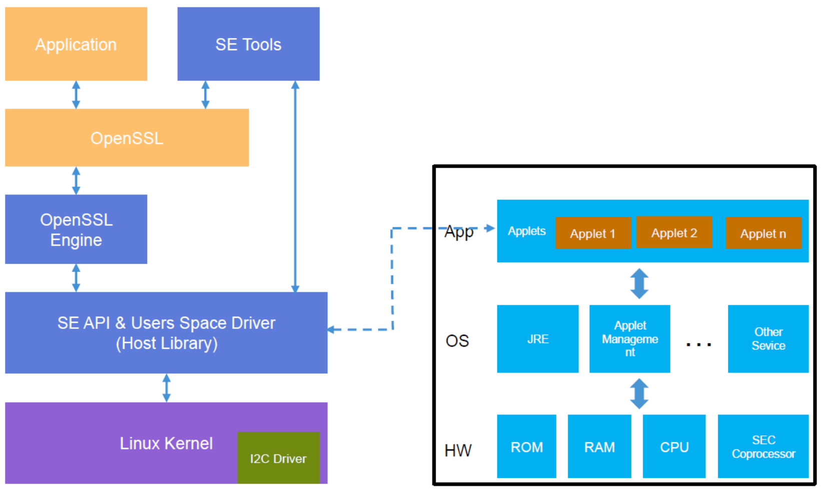
AES_ECB or AES_CCM encryption using a 256-bit key.

Trusted descriptors provide a means for trustworthy software to create trusted “applets”, that can be safely executed by less trustworthy software.

SEC ensures the integrity of the trusted descriptor with a cryptographic signature. When executing the trusted descriptor, that black key is not decrypted unless the signature is valid.

Trusted descriptors can be created only by trusted software.

2. Trust Software Solution on Layerscape platforms



3. Application development with OpenSSL Engine to offload encrypt/decrypt

This section provides examples of usage with OpenSSL Engine to offload encrypt/decrypt on hardware secure engine. OpenSSL provides the support of engine(basically hardware devices) to store the keys on the hardware devices to make keys more secure. This demo using the OpenSSL APIs to access the secure objects library.

Proprietary interfaces using Secure Object Library are provided to interact with the hardware for generating key pair within the Hardware security modules. Installing existing key in the Hardware security modules. Manufacturing Protection key operations. (MPKey)

The attached Secure Object Library based OpenSSL Engine that is used to communicate with underlying hardware security modules.

It does following things:

RSA Private Encryption. (RSA_Private_Encrypt)

RSA Private Decryption. (RSA_Private_Decrypt)

All other RSA operations will be done by OpenSSL itself.

Usage with OpenSSL using the engine from command Line.

Change the following in openssl.cnf (often in /etc/ssl/openssl.cnf).

This line must be placed at the top, before any sections are defined:

openssl_conf = conf_section

Add following section at bottom of file:

```
[conf_section]

engines = engine_section

[engine_section]

secure_obj = sobj_section

[sobj_section]

engine_id = eng_secure_obj

dynamic_path = <path where lib_eng_secure_obj.so is placed>

default_algorithms = RSA

init = 1
```

Testing the engine operation:

To verify that the engine is properly operating, you can use the following.

```
~# openssl engine

(dynamic) Dynamic engine loading support

(eng_secure_obj) secure object OpenSSL Engine.
```

If you do not update the OpenSSL configuration file, specify the engine configuration explicitly.

```
~# openssl engine -t dynamic -pre SO_PATH:<path-to-libeng_secure_obj.so>

-pre ID:eng_secure_obj -pre LIST_ADD:1 -pre LOAD

  (dynamic) Dynamic engine loading support

[Success] : SO_PATH:/usr/lib/aarch64-linux-gnu/openssl-1.0.0/engines/libeng_secure_obj.SO

[Success] : ID:eng_secure_obj

[Success] : LIST_ADD:1

[Success] : LOAD
```

```
LOADED: (eng_secure_obj) Secure object OpenSSL Engine.

        [available]
```

As per the following examples, generate a private key in the hardware security modules with sobj_app, This will also create a fake PEM file “dev_key.pem” having information to get the required key from hardware security modules.

```
$: sobj_app -G -m rsa-pair -s 2048 -l "Test_Key" -i 1 -w dev_key.pem
```

To generate a certificate with key in the Secure Object module, the following commands can be used:

```
$ openssl req -new -key dev_key.pem -out req.pem -text -x509 -subj "/CN=NXP"

$ openssl x509 -signkey dev_key.pem -in req.pem -out cert.pem
```

The first command creates a self-signed Certificate for “NXP ". The signing is done using the key specified by the fake PEM file.

The second command creates a self-signed certificate for the request, the private key used to sign the certificate is the same private key used to create the request.