Linux CE
开发指南

版本号: 2.9
发布日期: 2025.06.27

版本历史

版本号	日期	制/修订人	内容描述
1.0	2020.7.19	AWA0480	初始版本。
2.0	2021.04.02	AWA0480	适配 linux-5.4 平台。
2.1	2023.04.02	XAA0193	1. 对术语部分的格式进行了调整。2. 对 openssl 的编译使用章节进行了详细的描述。3. 对部分目录结构进行了调整使其更加清晰明了。
2.2	2023.08.02	XAA0248	适配 bsp 仓库。
2.3	2023.09.28	XAA0248	更新适用产品信息。
2.4	2024.09.18	XAA0175	更新适用产品信息，增加 H618。
2.5	2024.10.21	XAA0190	更新适用产品信息, 增加 MR536, T536。
2.6	2024.11.23	XAA0309	更新适用产品信息，增加 A733。
2.7	2025.02.20	XAA0309	更新内核版本与适用产品信息，增加 H135。
2.8	2025.02.21	XAA0345	1. 对部分目录结构进行了调整 2. 更新内核版本与适用产品信息，增加 A537 和 A333。3. 新增 hwrng 调用的配置方式 4. 完善 openssl 的编译方法 5. 修改算法接口参数、结构体信息
2.9	2025.06.27	XAA0345	1. 更新内核版本与适用产品信息，增加 MR153 和 T153。

目　录

1 概述 **1**

1.1 编写目的 . . . 1
1.2 适用范围 . . . 1
1.3 相关人员 . . . 2
1.4 相关术语 . . . 2

2 CE 模块描述 **3**

2.1 CE 的算法支持 . . . 3
2.2 Linux Crypto 算法框架 . . . 3
2.2.1 使用 openssl 方式 . . . 4
2.2.2 CE 设备节点方式 . . . 5
2.3 模块配置介绍 . . . 6
2.3.1 加解密接口的选择 . . . 6
2.3.1.1 选择 Linux 内核源生的配置 . . . 6
2.3.1.2 选择 ARM 的加速指令的配置 . . . 7
2.3.1.3 选择 CE openssl 调用方式配置 . . . 10
2.3.1.4 选择 CE 设备节点调用方式配置 . . . 12
2.3.1.5 选择 CE hwrng 算法调用方式配置 . . . 13
2.3.2 Device Tree 配置说明 . . . 13
2.4 源码结构介绍 . . . 14
2.4.1 bsp 独立仓库源代码结构 . . . 14

3 模块接口描述 **16**

3.1 算法注册接口 . . . 16
3.1.1 crypto_register_alg() . . . 16
3.1.1.1 函数原型 . . . 16
3.1.1.2 功能描述 . . . 16
3.1.1.3 返回值 . . . 16
3.1.1.4 参数说明 . . . 16
3.1.2 crypto_unregister_alg() . . . 17
3.1.2.1 函数原型 . . . 17
3.1.2.2 功能描述 . . . 17
3.1.2.3 返回值 . . . 17
3.1.2.4 参数说明 . . . 17
3.1.3 crypto_register_ahash() . . . 18
3.1.3.1 函数原型 . . . 18
3.1.3.2 功能描述 . . . 18
3.1.3.3 返回值 . . . 18
3.1.3.4 参数说明 . . . 18

3.1.4 crypto_unregister_ahash() . . . 18
3.1.4.1 函数原型 . . . 18
3.1.4.2 功能描述 . . . 18
3.1.4.3 返回值 . . . 18
3.1.4.4 参数说明 . . . 19
3.2 算法处理接口 . . . 19
3.2.1 ss_aes_start() . . . 19
3.2.1.1 函数原型 . . . 19
3.2.1.2 功能描述 . . . 19
3.2.1.3 返回值 . . . 19
3.2.1.4 参数说明 . . . 19
3.2.2 ss_hash_start . . . 20
3.2.2.1 函数原型 . . . 20
3.2.2.2 功能描述 . . . 20
3.2.2.3 返回值 . . . 21
3.2.2.4 参数说明 . . . 21
3.2.3 ss_rng_start() . . . 21
3.2.3.1 函数原型 . . . 21
3.2.3.2 功能描述 . . . 21
3.2.3.3 返回值 . . . 21
3.2.3.4 参数说明 . . . 22

4 openssl 方式使用说明 23
4.1 基本介绍 . . . 23
4.2 openssl 的编译 . . . 24
4.3 openssl 调用方式的 demo 用例说明 . . . 25
4.3.1 使用 af_alg 引擎 . . . 25
4.3.2 MD5 demo . . . 26
4.3.3 AES demo . . . 27
4.3.4 HMAC-SHA1 demo . . . 29
4.3.5 DH demo . . . 31

5 CE 设备节点方式使用说明 35
5.1 AES demo . . . 35
5.2 DES/3DES demo . . . 40
5.3 HASH demo . . . 40
5.4 PRNG demo . . . 43
5.5 CRC demo . . . 45

6 Linux CRYPTO API 使用说明 48
6.1 hash 接口 . . . 48

插　图

图 2-1　Linux Crypto 算法框架图 3
图 2-2　openssl 调用方式图 4
图 2-3　CE 设备节点调用方式图 5
图 2-4　Cryptographic API 配置 6
图 2-5　Cryptographic API 配置 7
图 2-6　arm 加速指令配置 7
图 2-7　aes 类算法 neon 加速 8
图 2-8　net 选项 8
图 2-9　networking_options 9
图 2-10　sockets 接口 9
图 2-11　Userspace_interface 10
图 2-12　关闭 ce 硬件接口 10
图 2-13　AW_CE_SOCKET 11
图 2-14　Userspace_interface 11
图 2-15　net 选项 11
图 2-16　networking_options 12
图 2-17　sockets 接口 12
图 2-18　AW_CE_IOCTL 13

1 概述

1.1 编写目的

本文档对 Sunxi 平台 CE 硬件的加密和解密功能接口使用进行详细的阐述，让用户明确掌握加解密接口的编程方法。

1.2 适用范围

表 1-1: 适用内核版本列表

内核版本	驱动文件
Linux-6.6	bsp/drivers/ce/
Linux-6.6-xuantie	bsp/drivers/ce/
Linux-5.15	bsp/drivers/ce/
Linux-5.15-origin	bsp/drivers/ce/
Linux-5.4-ansc	bsp/drivers/ce/

表 1-2: 适用产品版本说明列表

芯片代号	CE 版本	内核版本
A523	V5	5.15
A527	V5	5.15
T527	V5	5.15
MR527	V5	5.15
AI985	V5	5.15
H618	V3	5.15
V821	V1	5.4-ansc
MR536	V5	5.15
T536	V5	5.15
A733	V5	6.6
H135	V3	6.6-xuantie
A537	V5	6.6
A333	V5	6.6

芯片代号	CE 版本	内核版本
MR153	V5	5.15-origin
T153	V5	5.15-origin

1.3 相关人员

CE 驱动、加解密应用层的开发/维护人员。

1.4 相关术语

术语	详细介绍
3DES	3DES 基于 DES 的一种改进算法，它使用 3 条 64 位的密钥对数据进行三次加密
AES	Advanced Encryption Standard，高级加密标准
API	Application Program Interface 应用程序接口
CBC	Cipher Block Chaining mode，加密块链模式
CE	Crypto Engine，算法引擎，以前称作 SS
CFB	Cipher feedback，密码反馈模式
CRC32:	Cyclic redundancy check 32，循环冗余校验（32 位）
CTR	Counter mode，计数模式
CTS	Ciphertext Stealing mode
DES	Data Encryption Standard，数据加密标准
DH	Diffie-Hellman 算法，密码一致协议
ECB	Electronic Code Book mode，电子密码本模式
ECC	Elliptic curve cryptography，椭圆曲线加密算法
ECDH	EC-based DH，基于椭圆曲线的密码交换协议
HMAC	Hash-based Message Authentication Code，基于 Hash 的消息鉴别码
HMAC-SHA1	SHA1-based HMAC，基于 SHA1 的 HMAC 算法
HMAC-SHA256	SHA256-based HMAC，基于 SHA256 的 HMAC 算法
MD5	Message Digest Algorithm 5，消息摘要算法第五版
OFB	Output feedback，输出反馈模式
PRNG	Pseudo Random Number Generator，伪随机数发生器
RSA	公钥加密算法
SHA	Secure Hash Algorithm，安全散列算法
SS	Security System，Sunxi SOC 中的系统安全模块，支持多种硬件加密解密算法
SUNXI	指 Allwinner 的一系列 SOC 硬件平台
TRNG	True Random Number Generator, 真随机数发生器
XTS	XEX-based tweaked-codebook mode with ciphertext stealing

2 CE 模块描述

2.1 CE 的算法支持

由于不同 sunxi 平台，硬件 CE 支持的算法不一样，因此需要了解支持具体的算法类型，请查阅相关平台 User Manual 的 CE 章节。

2.2 Linux Crypto 算法框架

Crypto 是内核一个独立的子系统，源码在 kernel/crypto 下，它实现了对算法的统一管理，并提供出统一的数据处理接口给其他子系统使用；因此基于这套框架，我们不仅可以使用 kernel 已有的 crypto 算法对数据做转换，还能自行扩展添加算法，整个算法框架如下。

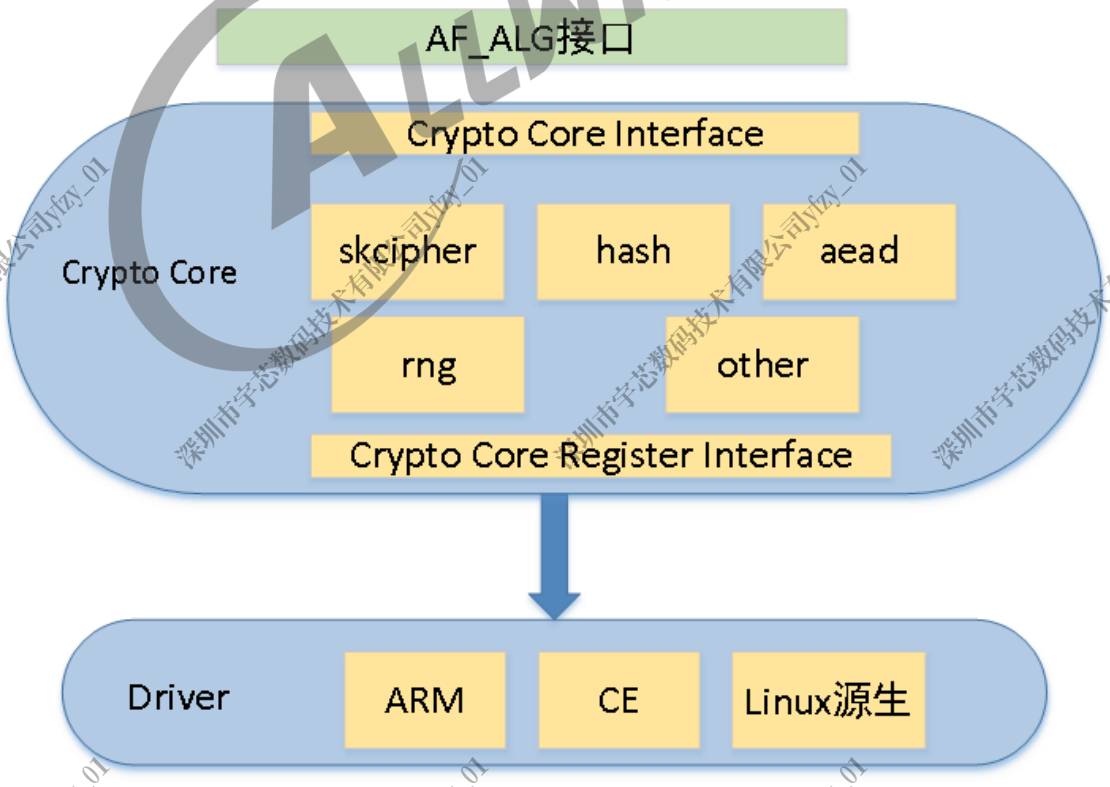

图 2-1: Linux Crypto 算法框架图

它实现了对称加解密，非对称加解密，认证加解密，hash，Hmac，伪随机数生成算法和压缩算法。

2.2.1 使用 openssl 方式

CE 按照 Linux 内核中的 crypto 框架设计，在应用层能够和 OpenSSL 完美配合，很容易扩展完成多种硬件算法的支持。整个软件架构的关系图如下:

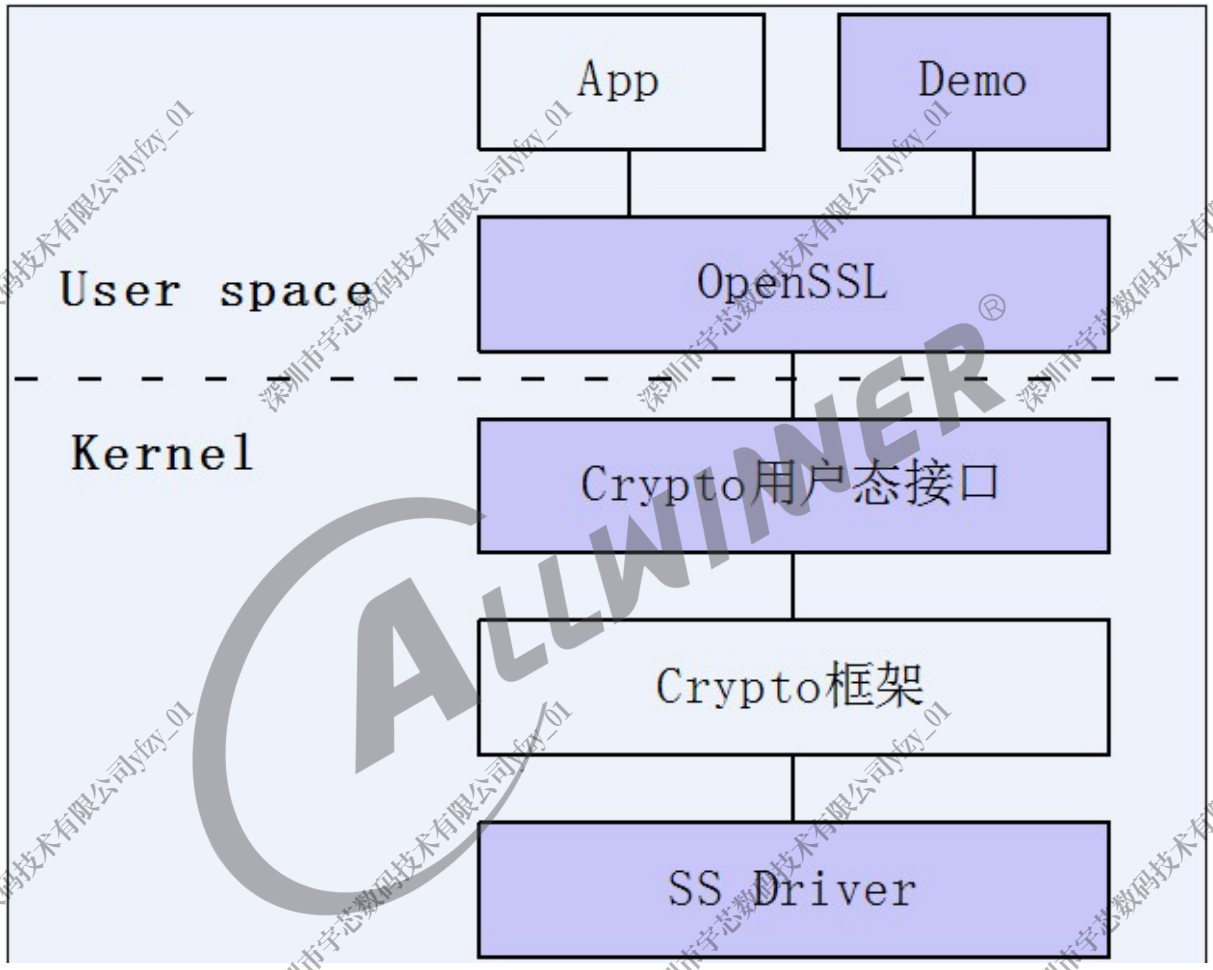

图 2-2: openssl 调用方式图

其中，[App] 是指用户的应用程序；[Crypto 框架] 是 Linux 内核自带的加密算法管理框架；紫色区域需要我们开发或修改，它们分别是：

1. Demo，基于 OpenSSL 的示例代码。
2. OpenSSL，一个基于密码学的安全开发包，OpenSSL 提供的功能相当强大和全面。
3. Crypto 用户态接口，内核 crypto 框架和用户态的接口部分。
4. SS Driver 即 CE Driver，负责操作 CE 硬件控制器。

可以看到，和用户应用程序直接打交道的是 OpenSSL 标准接口（将在第 4 章详述），这样 App 也很容易嵌入硬件的加解密功能。需要指出，标准的 OpenSSL 还不能直接和内核中的 Crypto 框架互通，需要在 OpenSSL 中注册一个引擎插件 (af_alg 插件)，并在 App 中要配置 OpenSSL 使用 af_alg 引擎。（使用方法详见 Demo）。

2.2.2 CE 设备节点方式

由于某些应用场景中，不想使用 OpenSSL 标准接口来操作 CE 的接口，因为 openssl 编译出来的库比较大，不适合小内存方案。因此 CE 驱动还提供 CE 设备节点方式供用户空间使用，整个软件架构的关系图如下:

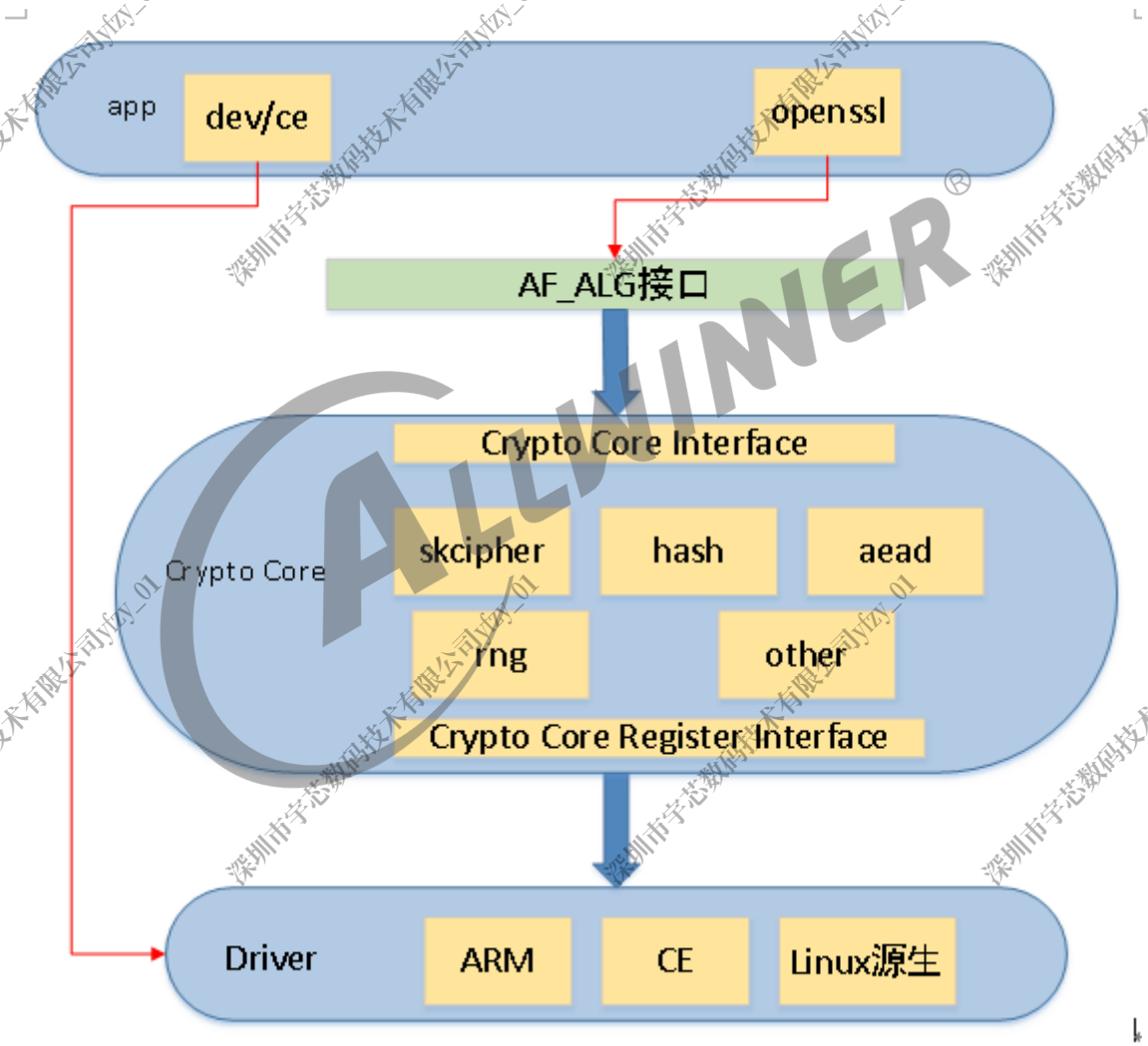

图 2-3: CE 设备节点调用方式图

如图所示，通过 CE 的设备节点方式不经过 Crypto 的框架，直接调用加解密接口。

2.3 模块配置介绍

在 ${SDK} 目录下执行：./build.sh menuconfig 进入配置主界面，并按以下步骤操作。

2.3.1 加解密接口的选择

Linux 内核支持 3 种加解密接口:

加解密接口	备注
Linux 内核源生加解密接口	C 语言实现
ARM 加解密接口	采用 ARM 的加速指令实现
CE 加解密接口	加解密硬件加速模块

2.3.1.1 选择 Linux 内核源生的配置

1. 首先选择 Cryptographic API 选项进入下一级配置，如图所示：

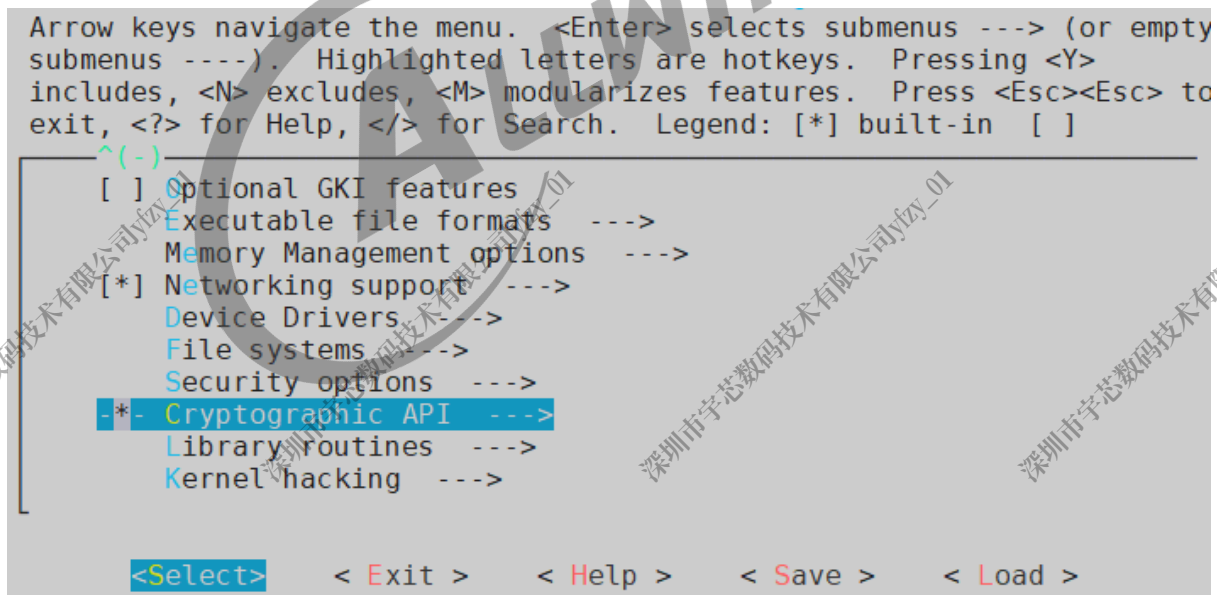

图 2-4: Cryptographic API 配置

2. 在 Cryptographic API 中选择相应的算法即可。

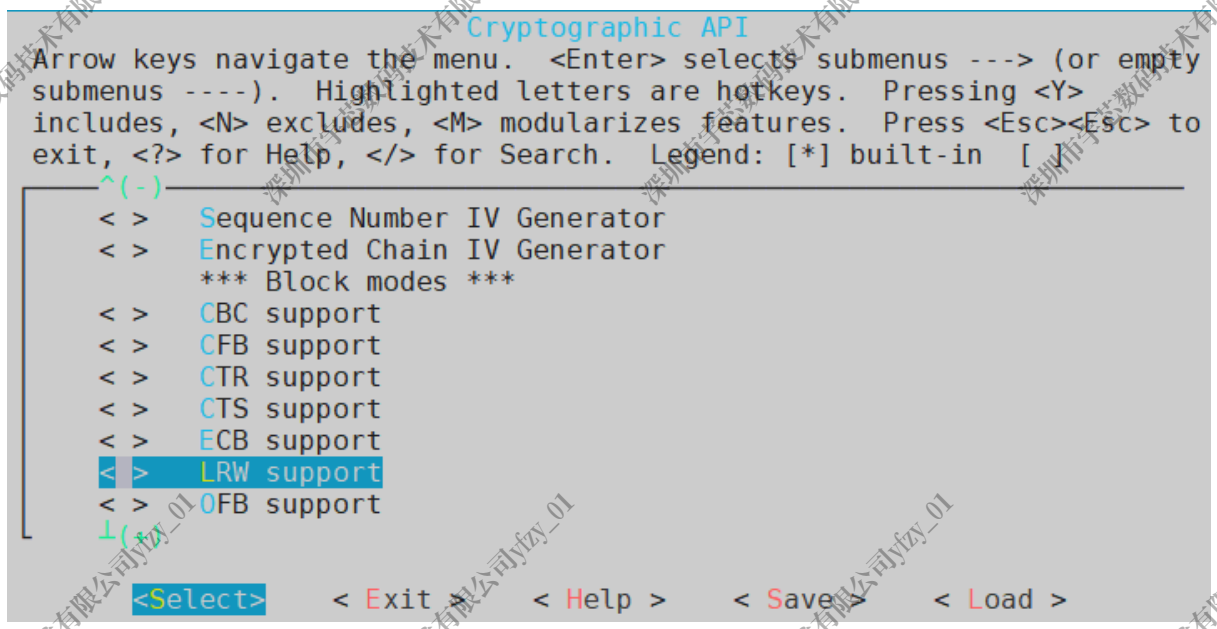

图 2-5: Cryptographic API 配置

2.3.1.2 选择 ARM 的加速指令的配置

如果数据块是以 8K 为单位，或 8K 以下，可以采用 ARM 的加速指令这比 CE 模块的性能更加有优势，需要注意的是如果开启 ARM 的加速指令，必须关闭 CE 的配置，因为 CE 的配置优先级更加高。

这里以 ARM-V8 架构的加速指令的进行配置：

1. 在 SDK 根目录下执行执行如下命令进行 menuconfig 配置：

```
SDK$ ./build.sh menuconfig
```

2. 选择 arm 加速指令配置。

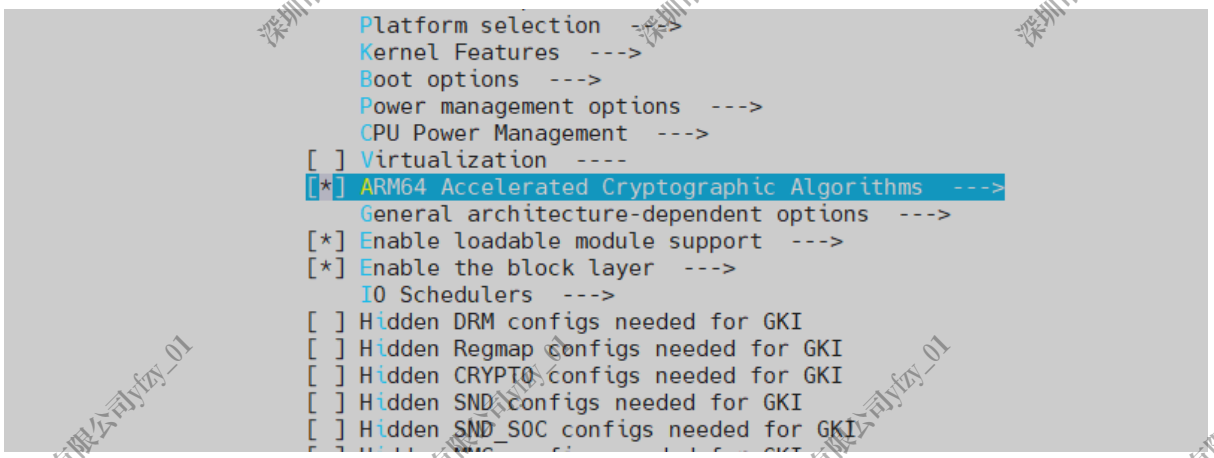

图 2-6: arm 加速指令配置

3. 选择 aes 类算法 neon 加速。

```
< >   SHA-384/SHA-512 digest algorithm (ARMv8 Crypto Extensions)
< >   SHA3 digest algorithm (ARMv8.2 Crypto Extensions)
< >   SM3 digest algorithm (ARMv8.2 Crypto Extensions)
< >   SM4 symmetric cipher (ARMv8.2 Crypto Extensions)
< >   GHASH/AES-GCM using ARMv8 Crypto Extensions
< >   POLYVAL using ARMv8 Crypto Extensions (for HCTR2)
< >   AES core cipher using scalar instructions
< >   AES core cipher using ARMv8 Crypto Extensions
< >   AES in CCM mode using ARMv8 Crypto Extensions
< >   AES in ECB/CBC/CTR/XTS/XCTR modes using ARMv8 Crypto Extensions
<*>   AES in ECB/CBC/CTR/XTS/XCTR modes using NEON instructions
< >   ChaCha20, XChaCha20, and XChaCha12 stream ciphers using NEON instructions
< >   Poly1305 hash function using scalar or NEON instructions
< >   NHPoly1305 hash function using NEON instructions (for Adiantum)
< >   AES in ECB/CBC/CTR/XTS modes using bit-sliced NEON algorithm
```

图 2-7: aes 类算法 neon 加速

4. 打开 net。

```
    Executable file formats  --->
    Memory Management options  --->
[*] Networking support  --->
    Device Drivers  --->
    File systems  --->
    Security options  --->
-*- Cryptographic API  --->
    Library routines  --->
```

图 2-8: net 选项

5. 选择 networking options 选项。

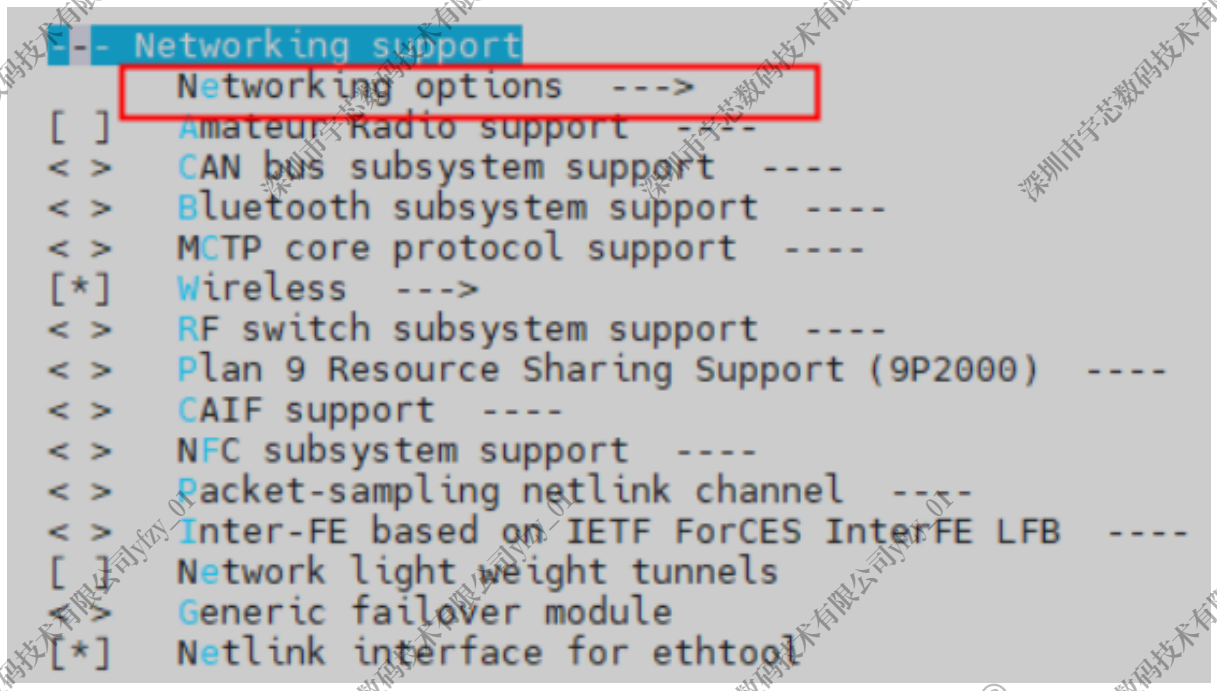

图 2-9: networking_options

6. 打开 sockets 接口。

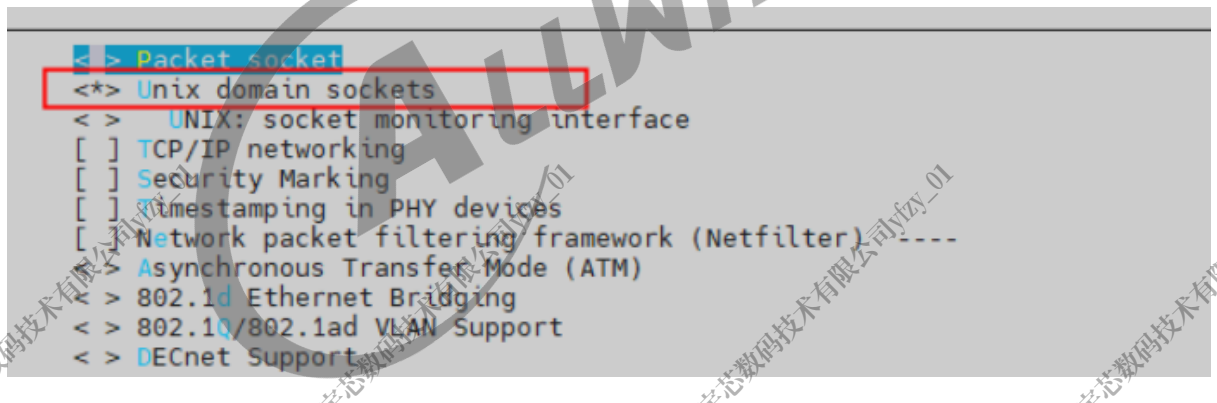

图 2-10: sockets 接口

7. 打开用户接口。

选择并进入Cryptographic/Userspace interface，选择 crypto 的接口配置：

CONFIG_CRYPTO_USER_API。

CONFIG_CRYPTO_USER_API_HASH。

CONFIG_CRYPTO_USER_API_SKCIPHER。

CONFIG_CRYPTO_USER_API_RNG。

CONFIG_CRYPTO_USER_API_AEAD。

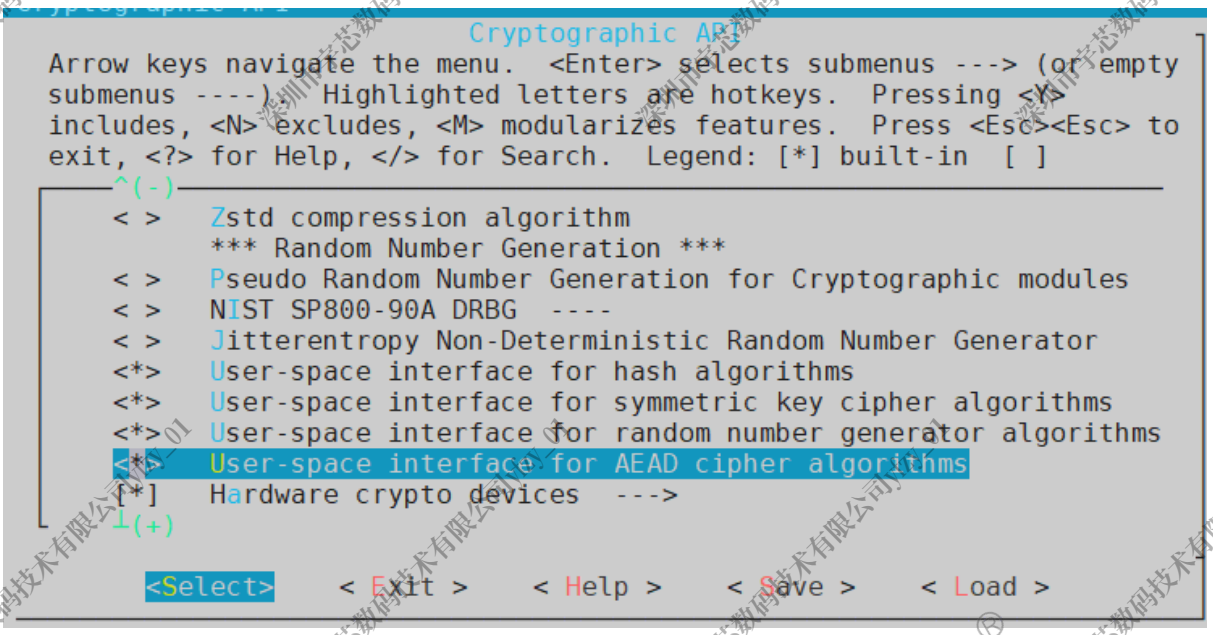

图 2-11: Userspace_interface

8. 关闭 ce 硬件接口。

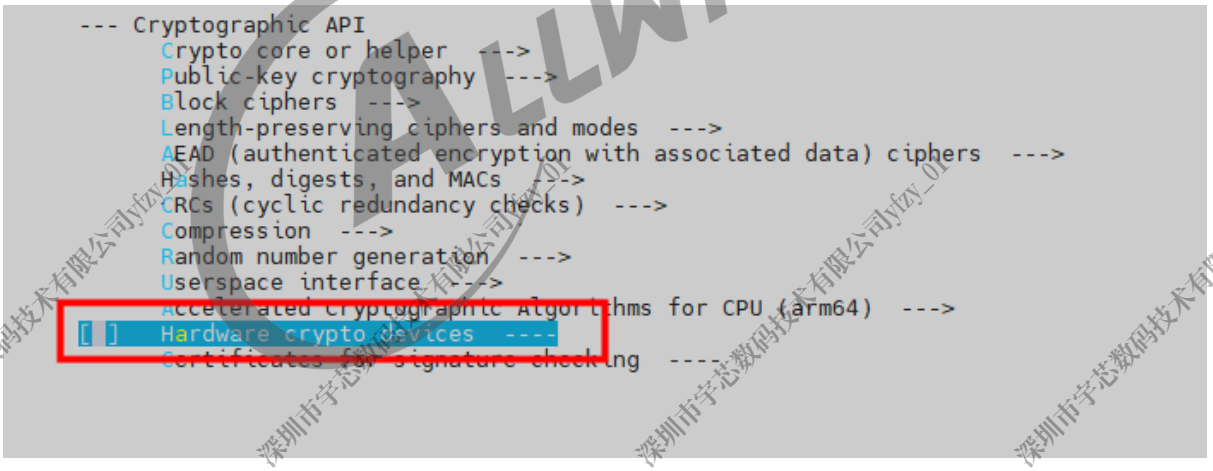

图 2-12: 关闭 ce 硬件接口

2.3.1.3 选择 CE openssl 调用方式配置

1. 在 SDK 根目录下执行执行如下命令进行 menuconfig 配置：

```
SDK$ ./build.sh menuconfig
```

2. 打开 CE 配置：CONFIG_AW_CE_SOCKET，可选择直接编译进内核，也可编译成模块。如下图所示：

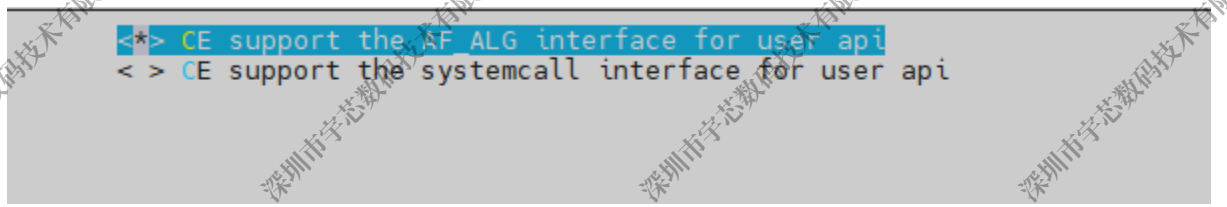

图 2-13: AW_CE_SOCKET

3. 选择并进入Cryptographic/Userspace interface，选择 crypto 的接口配置：CONFIG_CRYPTO_USER_API、CONFIG_CRYPTO_USER_API_HASH、CONFIG_CRYPTO_USER_API_SKCIPHER、CONFIG_CRYPTO_USER_API_RNG、CONFIG_CRYPTO_USER_API_AEAD。

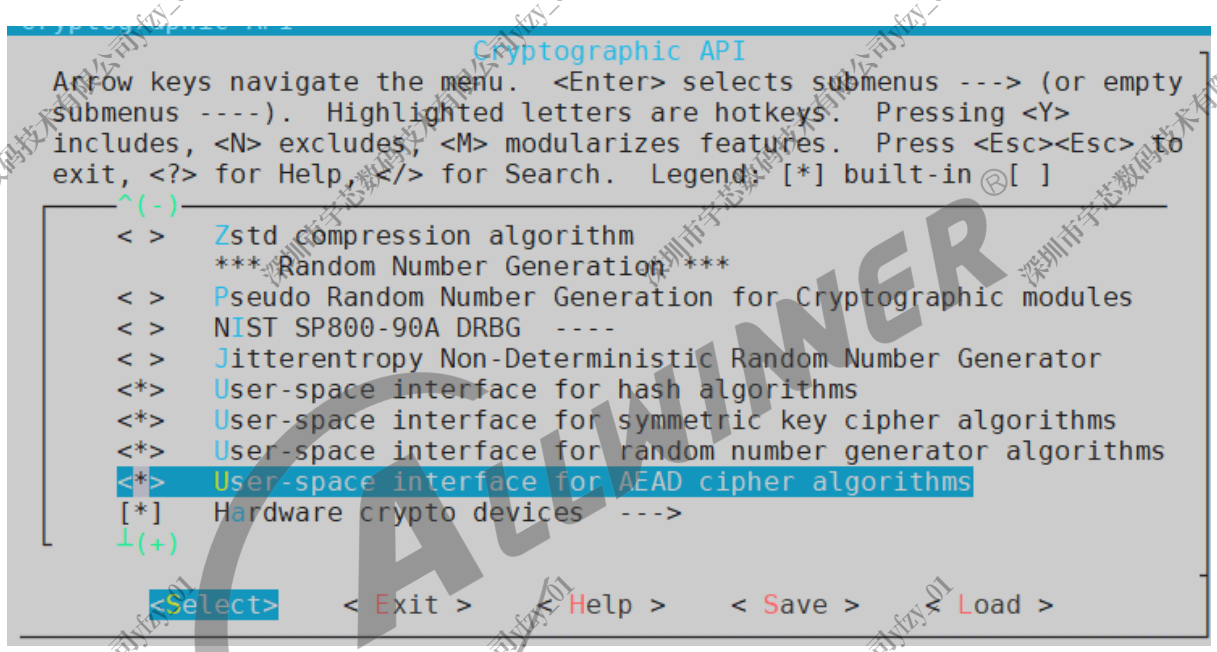

图 2-14: Userspace_interface

4. 打开 net。

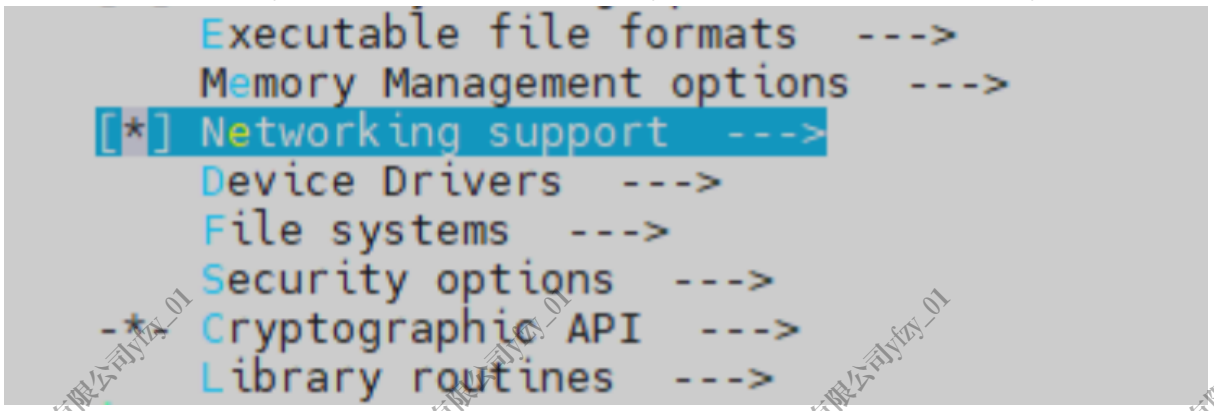

图 2-15: net 选项

5. 选择 networking options 选项。

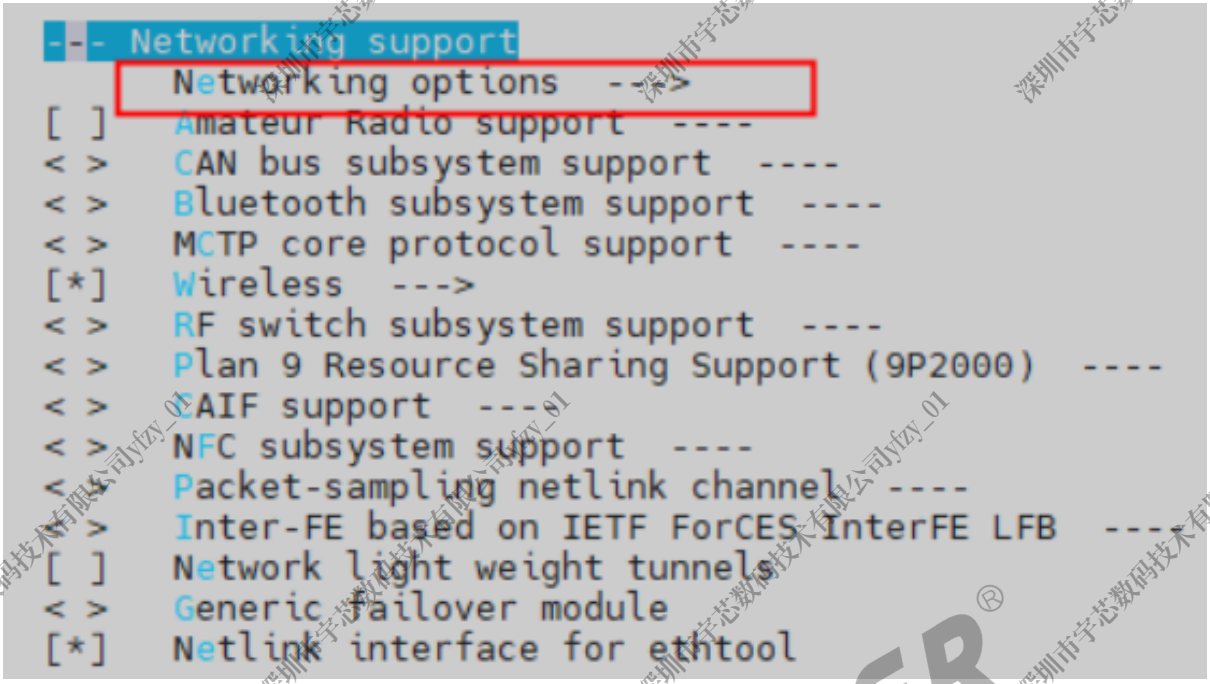

图 2-16: networking_options

6. 打开 sockets 接口。

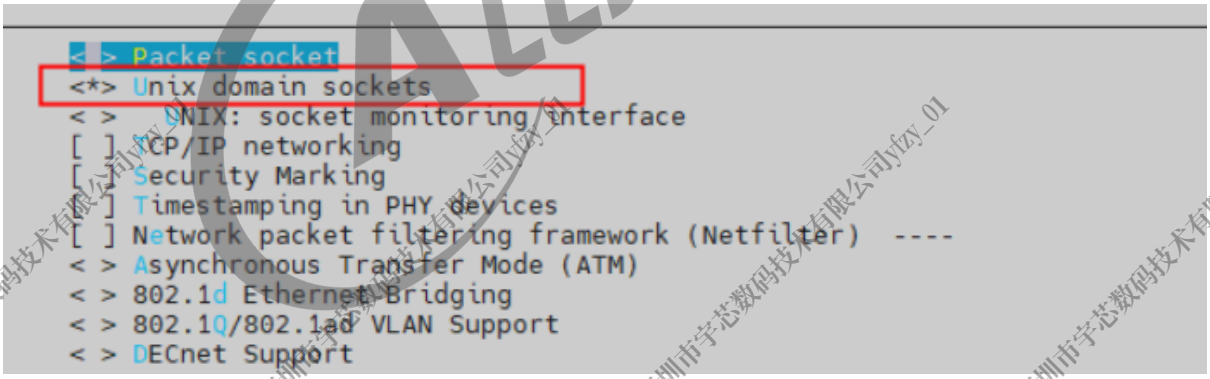

图 2-17: sockets 接口

2.3.1.4 选择 CE 设备节点调用方式配置

1. 在 SDK 根目录下执行执行如下命令进行 menuconfig 配置：

```
SDK$ ./build.sh menuconfig
```

2. 打开 CE 配置：CONFIG_AW_CE_IOCTL，可选择直接编译进内核，也可编译成模块，CE 设备节点调用方式配置与 openssl 调用方式配置只能二选一。如下图所示：

```
< > CE support the AF_ALG interface for user api
<*> CE support the systemcall interface for user api
```

图 2-18: AW_CE_IOCTL

2.3.1.5 选择 CE hwrng 算法调用方式配置

1. 在 SDK 根目录下执行执行如下命令进行 menuconfig 配置：

```
SDK$ ./build.sh menuconfig
```

2. 打开 CE 配置：

(1) 在打开 CE 设备节点配置或者 openssl 配置的基础上进行配置：hwrng 算法依赖于硬件随机数配置，需要先配置硬件随机数：CONFIG_HW_RANDOM，选择直接编译进内核。如下所示：

```
Device Drivers  --->
  Character devices  --->
    <*> Hardware Random Number Generator Core support
```

(2) 打开 hwrng 配置：CONFIG_AW_HWRNG_DRIVER，选择直接编译进内核。如下所示：

```
Allwinner BSP  --->
  Device Drivers  --->
    CE Drivers  --->
      <*> Support for sunxi hwrng driver
```

2.3.2 Device Tree 配置说明

在 Device Tree 中对 CE 控制器进行配置, 如下:

```
cryptoengine: ce@3040000 {
  compatible = "allwinner,sunxi-ce";
  device_name = "ce";
  reg = <0x0 0x03040000 0x0 0xa0>, /* non-secure space */
    <0x0 0x03040800 0x0 0xa0>; /* secure space */
  interrupts = <GIC_SPI 52 IRQ_TYPE_EDGE_RISING>, /*non-secure*/
    <GIC_SPI 53 IRQ_TYPE_EDGE_RISING>; /* secure*/
  clock-frequency = <400000000>; /* 400MHz */
  clocks = <&ccu CLK_BUS_CE>, <&ccu CLK_CE>, <&ccu CLK_CE_MBUS_GATE>, <&ccu CLK_PLL_PERI0_400M>;
  clock-names = "bus_ce", "ce_clk", "mbus_ce", "clk_src";
  resets = <&ccu RST_BUS_CE>;
};
```

其中:

1. compatible: 表征具体的设备, 用于驱动和设备的绑定。
2. reg: 设备使用的地址。
3. interrupts: 设备使用的中断。
4. clock-frequency: 设备使用的时钟频率。
5. clocks: 设备使用的时钟源。

2.4 源码结构介绍

2.4.1 bsp 独立仓库源代码结构

CE 驱动的源代码位于内核在 bsp/drivers/ce 目录下。

```
bsp/drivers/ce$ tree
├── Kconfig
├── Makefile
├── sunxi_ce.c          // Sunxi平台CE算法注册、处理流程的实现--硬件资源申请和使能
├── sunxi_ce_cdev.c     // CE的字符设备相关注册实现--硬件资源申请和使能，ioctl逻辑实现
├── sunxi_ce_cdev.h     // CE字符设备相关头文件
├── sunxi_hwrng.c       // CE的hwrng算法实现
├── sunxi_ce_proc_comm.c    // Sunxi平台CE算法处理流程的实现--socket逻辑实现
├── sunxi_ce_proc.h     // Sunxi平台CE算法相关头文件
├── v1
│   ├── sunxi_ce_cdev_comm.c    // V1版本SS控制器的算法处理过程
│   ├── sunxi_ce_proc.c         // V1CE器的算法处理过程
│   ├── sunxi_ce_reg.c          // V1版本SS控制器的寄存器接口实现
│   └── sunxi_ce_reg.h          // V1版本SS控制器的寄存器接口声明
├── v2
│   ├── sunxi_ce_proc.c
│   ├── sunxi_ce_reg.c
│   └── sunxi_ce_reg.h
├── v3
│   ├── sunxi_ce_cdev_comm.c
│   ├── sunxi_ce_proc.c
│   ├── sunxi_ce_proc_walk.c
│   ├── sunxi_ce_reg.c
│   └── sunxi_ce_reg.h
├── v4
│   ├── sunxi_ce_cdev_comm.c
│   ├── sunxi_ce_proc.c
│   ├── sunxi_ce_reg.c
│   └── sunxi_ce_reg.h
└── v5
    ├── sunxi_ce_cdev_comm.c
    ├── sunxi_ce_proc.c
    ├── sunxi_ce_reg.c
    └── sunxi_ce_reg.h
```

通过 Makefile 控制五种 CE 版本的源码编译，Makefile 内容如下:

```
# SPDX-License-Identifier: GPL-2.0
#
# Makefile for the ce drivers.
#
```

```
obj-$(CONFIG_AW_CE_SOCKET) += sunxi-ce.o
obj-$(CONFIG_AW_CE_IOCTL) += sunxi-ce-ioctl.o

sunxi-ce-ioctl-$(CONFIG_AW_CE_IOCTL) += sunxi_ce_cdev.o
sunxi-ce-ioctl-$(CONFIG_AW_HWRNG_DRIVER) += sunxi_hwrng.o
sunxi-ce-$(CONFIG_AW_CE_SOCKET) += sunxi_ce.o sunxi_ce_proc_comm.o
sunxi-ce-$(CONFIG_AW_HWRNG_DRIVER) += sunxi_hwrng.o

ifdef CONFIG_ARCH_SUN300IW1
  AW_CE_VER = v1
endif
ifdef CONFIG_ARCH_SUN8IW20
  AW_CE_VER = v3
endif
ifdef CONFIG_ARCH_SUN50IW9
  AW_CE_VER = v3
endif
ifdef CONFIG_ARCH_SUN251IW1
  AW_CE_VER = v3
endif
ifdef CONFIG_ARCH_SUN50IW10
  AW_CE_VER = v4
endif
ifdef CONFIG_ARCH_SUN55IW3
  AW_CE_VER = v5
endif
ifdef CONFIG_ARCH_SUN55IW6
  AW_CE_VER = v5
endif
ifdef CONFIG_ARCH_SUN8IW21
  AW_CE_VER = v5
endif
ifdef CONFIG_ARCH_SUN65IW1
  AW_CE_VER = v5
endif
ifdef CONFIG_ARCH_SUN60IW2
  AW_CE_VER = v5
endif

sunxi-ce-$(CONFIG_AW_CE_SOCKET) += $(AW_CE_VER)/sunxi_ce_reg.o $(AW_CE_VER)/sunxi_ce_proc.o
sunxi-ce-ioctl-$(CONFIG_AW_CE_IOCTL) += $(AW_CE_VER)/sunxi_ce_reg.o $(AW_CE_VER)/sunxi_ce_cdev_comm.o
obj-$(CONFIG_AW_TRNG) += sunxi_trng/sunxi_trng.o

obj-$(CONFIG_AW_CE_SOCKET_SECURE_KEY) += smc_process/sunxi_ce_smc_process.o
#ccflags-y += -I$(srctree)/drivers/crypto/sunxi-ce/$(AW_CE_VER)
#ccflags-y += -DDEBUG
```

3 模块接口描述

描述 CE 驱动涉及的对内、对外接口，只限于 Linux 内核范围内。

3.1 算法注册接口

以下接口都是 Linux 内核中 crypto 的标准接口，主要完成算法的注册、注销。

3.1.1 crypto_register_alg()

3.1.1.1 函数原型

```
int crypto_register_alg(struct crypto_alg *alg)
```

3.1.1.2 功能描述

向 crypto 框架注册一种加密算法。

3.1.1.3 返回值

返回 0 表示成功，返回其他值表示失败。

3.1.1.4 参数说明

alg，算法的一些描述、配置信息。

3.1.2 crypto_unregister_alg()

3.1.2.1 函数原型

```
int crypto_unregister_alg(struct crypto_alg *alg)
```

3.1.2.2 功能描述

从 crypto 框架注销一种加密算法。

3.1.2.3 返回值

返回 0 表示成功，返回其他值表示失败。

3.1.2.4 参数说明

alg，算法的一些描述、配置信息, 其中，结构 crypto_alg 的定义:

```
struct crypto_alg {
  struct list_head cra_list;
  struct list_head cra_users;

  u32 cra_flags;
  unsigned int cra_blocksize;
  unsigned int cra_ctxsize;
  unsigned int cra_alignmask;

  int cra_priority;
  atomic_t cra_refcnt;

  char cra_name[CRYPTO_MAX_ALG_NAME];
  char cra_driver_name[CRYPTO_MAX_ALG_NAME];

  const struct crypto_type *cra_type;

  union {
    struct ablkcipher_alg ablkcipher;
    struct aead_alg aead;
    struct blkcipher_alg blkcipher;
    struct cipher_alg cipher;
    struct compress_alg compress;
    struct rng_alg rng;
  } cra_u;

  int (*cra_init)(struct crypto_tfm *tfm);
  void (*cra_exit)(struct crypto_tfm *tfm);
  void (*cra_destroy)(struct crypto_alg *alg);
```

```
    struct module *cra_module;
};
```

3.1.3 crypto_register_ahash()

3.1.3.1 函数原型

```
int crypto_register_ahash(struct ahash_alg *alg)
```

3.1.3.2 功能描述

向 crypto 框架注册一种异步 Hash 类算法。

3.1.3.3 返回值

返回 0 表示成功，返回其他值表示失败。

3.1.3.4 参数说明

alg，算法的一些描述、配置信息。

3.1.4 crypto_unregister_ahash()

3.1.4.1 函数原型

```
int crypto_unregister_ahash(struct ahash_alg *alg)
```

3.1.4.2 功能描述

从 crypto 框注销一种异步 Hash 类算法。

3.1.4.3 返回值

返回 0 表示成功，返回其他值表示失败。

3.1.4.4 参数说明

alg，算法的一些描述、配置信息, 其中，结构 ahash_alg 的定义:

```
struct ahash_alg {
    int (*init)(struct ahash_request *req);
    int (*update)(struct ahash_request *req);
    int (*final)(struct ahash_request *req);
    int (*finup)(struct ahash_request *req);
    int (*digest)(struct ahash_request *req);
    int (*export)(struct ahash_request *req, void *out);
    int (*import)(struct ahash_request *req, const void *in);
    int (*setkey)(struct crypto_ahash *tfm, const u8 *key,
          unsigned int keylen);

    struct hash_alg_common halg;
};
```

3.2 算法处理接口

这里分 AES 类、Hash 类、RNG 类描述几种算法的核心处理函数接口，都是 SS 驱动内部的接口，它们通过控制 DMA、SS 控制器完成一次运算。

3.2.1 ss_aes_start()

3.2.1.1 函数原型

```
static int ss_aes_start(ss_aes_ctx_t *ctx, ss_aes_req_ctx_t *req_ctx, int len)
```

3.2.1.2 功能描述

执行一次 AES 类算法的运算。

3.2.1.3 返回值

0，成功；其他值，失败。

3.2.1.4 参数说明

1. ctx，AES 类算法的上下文。

2. req_ctx，一次 AES 类算法请求的上下文。
3. len，要计算的数据长度。

其中，ss_aes_ctx_t 的定义如下：

```
/* The common context of AES and HASH */
typedef struct {
    u32 flow;
    u32 flags;
} ss_comm_ctx_t;

typedef struct {
    ss_comm_ctx_t comm; /* must be in the front. */

#ifdef SS_RSA_ENABLE
    u8  key[SS_RSA_MAX_SIZE];
    u8  iv[SS_RSA_MAX_SIZE];
#else
    u8   key[AES_MAX_KEY_SIZE];
    u8   iv[AES_MAX_KEY_SIZE];
#endif
#ifdef SS_SCATTER_ENABLE
    u8  next_iv[AES_MAX_KEY_SIZE];   /* saved the next IV/Counter in continue mode */
#endif
    int key_size;
    int iv_size;
    int cnt;   /* in Byte */
} ss_aes_ctx_t;
```

ss_aes_req_ctx_t 的定义如下（源文件 sunxi_ss.h）

```
typedef struct {
    u32 dir;
    u32 type;
    u32 mode;
    u32 bitwidth;  /* the bitwidth of CFB mode */
    struct completion done;
    ss_dma_info_t dma_src;
    ss_dma_info_t dma_dst;
} ss_aes_req_ctx_t;
```

3.2.2 ss_hash_start

3.2.2.1 函数原型

```
u32 ss_hash_start(ss_hash_ctx_t *ctx, ss_aes_req_ctx_t *req_ctx, u32 len, u32 last)
```

3.2.2.2 功能描述

执行一次 HASH 类算法的运算。

3.2.2.3 返回值

返回 0 表示成功，返回其他值表示失败。

3.2.2.4 参数说明

1. ctx，Hash 类算法的上下文。
2. req_ctx，一次 AES 类算法请求的上下文。
3. len，要计算的数据长度。
4. last，判断是否是最后一次 hash 的标志位

其中，ss_hash_ctx_t 的定义如下

```
typedef struct {
    ss_comm_ctx_t comm; /* must be in the front. */
    u8  md[SS_DIGEST_SIZE];
    u8  pad[SS_HASH_PAD_SIZE];
    u32 tail_len;
    u32 md_size;
    u32 cnt;   /* in Byte */
    u32 npackets;
} ss_hash_ctx_t;
```

3.2.3 ss_rng_start()

3.2.3.1 函数原型

```
static int ss_rng_start(ss_aes_ctx_t *ctx, u8 *rdata, u32 dlen, u32 trng)
```

3.2.3.2 功能描述

执行一次 RNG 类算法的运算。

3.2.3.3 返回值

返回值大于 0，实际读取到的随机数长度；其他值，失败。

3.2.3.4 参数说明

1. ctx，RNG 类（和 AES 类共用同一种类型）算法的上下文。
2. rdata，用于保存输出的随机数。
3. dlen，要请求的随机数长度（字节为单位）。
4. trng，判断是 prng 算法还是 trng 算法的标志位

其中，ss_aes_ctx_t 的定义上述 aes 接口中有说明

4 openssl 方式使用说明

4.1 基本介绍

openssl 程序是一个命令行程序，用于从 shell 使用 OpenSSL **加密**库的各种加密函数。实现了安全套接字层 (SSL) 和传输层安全 (TLS) 网络协议以及它们所需的相关密码学标准。可同时对接不同的密码学实现，向用户层提供的 API 实现的一个软件包，可以通过命令行传参，进行多种不同的算法计算。

- openssl 的官方介绍：https://www.openssl.org/docs/manmaster/man1/openssl.html
- openssl 的官方使用介绍：https://opensource.com/article/19/6/cryptography-basics-openssl-part-2

OpenSSL 的接口说明, 可以在官网中找到对应的算法接口。下文以 demo 形式演示 OpenSSL 的几种应用,demo 源文件需要放在 OpenSSL 中，编译和运行都需要 OpenSSL 的动态库支持。

源码下载：

目前 openssl 已经集成到的 Longan 和 Tina5.0 SDK 中，分别位于如下位置：

```
tina5.0/platform/allwinner/openssl/openssl-1.0.0

${SDK}/platform/framework/openssl/openssl-1.0.0
```

如果你想单独下载，请使用如下链接：

```
git clone "ssh://${name}@gerrit.allwinnertech.com:29418/${SDK}/platfrom/framework/openssl/openssl-1.0.0"
```

源码结构：

```
openssl1.0.0
├── apps        # 所有应用程序代码的实现，就是使用Linux命令来进行ssl证书生成等操作的实现
├── Configurations # 配置文件
├── crypto      # 所有加解密算法，其中包括对称加密算法、非对称加密算法、哈希算法、随机数生成器等的实现。
├── demos       # demo代码
├── doc         # 一些函数的说明文档，可以看到cypto以及ssl目录均有相关说明文档
├── engines     # 所有engine实现的代码，用于控制管理各种加密算法操作，主要是方便硬件算法的加入
├── external
├── include     # 第三方程序调用openssl库的时候提供给第三方使用的接口头文件
├── out         # 相关库文件和可执行文件的生成目录
├── ssl         # 握手协议的实现，证书校验过程等，具体包括ssl3.0、tls1.0、tls1.1、tls1.2等主流版本
├── ss_test     # AW 自己的demo文件
├── test        # openssl自带原生的测试代码
├── tools
└── usr         # 新增配置完后，编译成功相关库文件
```

```
├── VMS
└── 其他
```

4.2 openssl 的编译

- 对于 Tina5.0 SDK，配置当前编译环境后进入如下 openssl 源码路径：

```
tina5.0$ ./build.sh config

$ tina5.0/platform/allwinner/openssl/openssl-1.0.0
```

- 对于 ${SDK} SDK，配置当前编译环境后进入如下 openssl 源码路径：

```
cd ${SDK}/platform/framework/openssl/openssl-1.0.0
```

- 执行如下命令完成编译 openssl 的编译：

```
openssl-1.0.0$ ./Configure  linux-elf --cross-compile-prefix={SDK}/out/toolchain/gcc-arm-10.3-2021.07-x86_64-aarch64-
      none-linux-gnu/bin/aarch64-none-linux-gnu- no-asm no-md2 no-mdc2 no-md4 no-rc4 no-camellia no-idea no-rc2
      no-bf no-cast no-srtp no-dsa no-hw-4758-cca no-hw-aep no-hw-atalla no-hw-cswift no-hw-ncipher no-hw-nuron
      no-hw-sureware no-hw-ubsec no-hw-padlock shared
其中：{SDK}代表当前环境的绝对路径
   {SDK}/out/toolchain/gcc-arm-10.3-2021.07-x86_64-aarch64-none-linux-gnu 代表编译工具链的地址
   aarch64-none-linux-gnu 代表交叉编译工具链
这两个参数都可以在tina目录下的.buildconfig文件中可以查询到，分别是``LICHEE_TOOLCHAIN_PATH``和``
      LICHEE_CROSS_COMPILER``
openssl-1.0.0$ make depend  #配置环境
openssl-1.0.0$ make clean  # 清除旧编译文件
openssl-1.0.0$ make       # 编译
openssl-1.0.0$ make install    # 生成库文件
```

- 编译完成后得到可执行文件 openssl，存放在：

```
openssl-1.0.0/out/usr/ssl/bin/openssl
```

- 编译完成的库文件存放在：

```
openssl-1.0.0/out/usr/ssl/lib/libcrypto.so.1.0.0      # 加解密运算的基本库

openssl-1.0.0/out/usr/ssl/lib/libssl.so.1.0.0       # 加解密运算的基本库

openssl-1.0.0/out/usr/ssl/lib/engines/libaf_alg.so     # 注册 af_alg engine相关的库
```

- openssl 的配置文件存放在：

```
openssl-1.0.0/out/usr/ssl/openssl.cnf
```

编译完成后需要将如下文件 cp 到小机端，供后续使用：

1. 将编译得到的 openssl 可执行文件 cp 到小机端
2. 将编译得到的 lib 库文件依次放到小机端的如下目录，应用层执行的时候需要链接这 3 个库文件：

```
/usr/lib/libcrypto.so.1.0.0

/usr/lib/libssl.so.1.0.0

/usr/ssl/lib/engines/libaf_alg.so
```

3. 将 openssl.cnf 配置文件 cp 到小机端的 /usr/ssl目录下。从文件在 opensll 标准调用时会使用到，而在下文中描述的 demo 用例方式中，并不会使用。

4.3 openssl 调用方式的 demo 用例说明

openssl 调用方式的 demo 用例全部存放在 openssl-1.0.0/ss_test/目录下。

1. 进入 openssl-1.0.0/ss_test 目录，执行 make 命令即可完成 demo 的编译：

```
openssl/openssl-1.0.0/ss_test$ make
```

2. 将编译的得到的 md5.bin 可执行文件 cp 到小机端，然后便可以进行测试。

4.3.1 使用 af_alg 引擎

因为要使用 af_alg 引擎，需要在初始化 OpenSSL 时显式的指定加密引擎。openssl_engine_init 函数负责完成此部分工作。

```
ENGINE *openssl_engine_init(char *type)
{
    ENGINE *e = NULL;
    const char *name = "af_alg";

    OpenSSL_add_all_algorithms();
    ENGINE_load_builtin_engines();

    e = ENGINE_by_id(name);
    if (!e) {
        DBG("find engine %s error\n", name);
        return NULL;
    }
```

```
    ENGINE_ctrl_cmd_string(e, "DIGESTS", type, 0);
    return e;
}

void openssl_engine_free(ENGINE *e)
{
    if (e != NULL)
        ENGINE_free(e);

    ENGINE_cleanup();
    EVP_cleanup();
}
```

4.3.2 MD5 demo

详细的 demo 文件请查看 openssl-1.0.0/ss_test/目录下。

```
void print_md(unsigned char *md, int len)
{
    int i;

    for (i=0; i<len; i++)
        printf("%02x", md[i]);
    printf("\n");
}

int main(int argc, char *argv[])
{
    int ret = 0;
    FILE *in = NULL;
    ENGINE *e = NULL;
    EVP_MD_CTX ctx = {0};
    const EVP_MD *e_md = NULL;
    unsigned int md_size = 0;
    unsigned char md[MD5_DIGEST_LENGTH] = {0};

    if (argc != PT_NUM) {
        usage();
        return -1;
    }

    in = fopen(argv[PT_IN_FILE], "rb");
    if (in == NULL) {
        DBG("Failed to fopen(%s)! \n", argv[PT_IN_FILE]);
        return -1;
    }

    e = openssl_engine_init("md5");
    if (e == NULL) {
        ret = -1;
        goto error;
    }

    e_md = ENGINE_get_digest(e, NID_md5);
    if (e_md == NULL) {
        DBG("ENGINE_get_digest() failed! \n");
```

```
    ret = -1;
    goto error;
  }

  EVP_DigestInit(&ctx, e_md);
  for (;;) {
    ret = fread(g_buf, 1, SS_TEST_BUF_SIZE, in);
    if (ret <= 0) {
      if (ret < 0)
        DBG("read(%d) return %d. \n", SS_TEST_BUF_SIZE, ret);
      break;
    }

    EVP_DigestUpdate(&ctx, g_buf, (unsigned long)ret);
  }
  EVP_DigestFinal(&ctx, md, &md_size);

  printf("MD5(%s)= ", argv[PT_IN_FILE]);
  print_md(md, MD5_DIGEST_LENGTH);

error:
  if (in != NULL)
    fclose(in);

  EVP_MD_CTX_cleanup(&ctx);
  openssl_engine_free(e);
  return ret;
}
```

4.3.3 AES demo

详细的 demo 文件请查看 openssl-1.0.0/ss_test/目录下。

```
/* The identification string to indicate the key source.*/
#define CE_KS_INPUT         "default"
#define CE_KS_SSK         "KEY_SEL_SSK"
#define CE_KS_HUK         "KEY_SEL_HUK"
#define CE_KS_RSSK         "KEY_SEL_RSSK"
#define CE_KS_INTERNAL_0   "KEY_SEL_INTRA_0"
#define CE_KS_INTERNAL_1   "KEY_SEL_INTRA_1"
#define CE_KS_INTERNAL_2   "KEY_SEL_INTRA_2"
#define CE_KS_INTERNAL_3   "KEY_SEL_INTRA_3"
#define CE_KS_INTERNAL_4   "KEY_SEL_INTRA_4"
#define CE_KS_INTERNAL_5   "KEY_SEL_INTRA_5"
#define CE_KS_INTERNAL_6   "KEY_SEL_INTRA_6"
#define CE_KS_INTERNAL_7   "KEY_SEL_INTRA_7"

unsigned char g_inbuf[SS_TEST_BUF_SIZE] = {0};
unsigned char g_outbuf[SS_TEST_BUF_SIZE] = {0};
unsigned char g_key[AES_KEY_SIZE_256] = {
        0xFF, 0xEE, 0xDD, 0xCC, 0xBB, 0xAA, 0x99, 0x88,
        0x77, 0x66, 0x55, 0x44, 0x33, 0x22, 0x11, 0x00,
        0x00, 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77,
        0x88, 0x99, 0xAA, 0xBB, 0xCC, 0xDD, 0xEE, 0xFF};
const unsigned char g_iv[AES_BLOCK_SIZE] = {
        0x00, 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77,
        0x88, 0x99, 0xAA, 0xBB, 0xCC, 0xDD, 0xEE, 0xFF};
```

```
int main(int argc, char *argv[])
{
    int ret = 0;
    int enc = 0;
    int inl = 0;
    int outl = 0;
    FILE *in = NULL;
    FILE *out = NULL;
    ENGINE *e = NULL;
    EVP_CIPHER_CTX ctx = {0};
    const EVP_CIPHER *e_cipher = NULL;

    if (argc != PT_NUM) {
        usage();
        return -1;
    }

    in = fopen(argv[PT_IN_FILE], "rb");
    if (in == NULL) {
        DBG("Failed to fopen(%s)! \n", argv[PT_IN_FILE]);
        return -1;
    }
    out = fopen(argv[PT_OUT_FILE], "wb");
    if (out == NULL) {
        DBG("Failed to fopen(%s)! \n", argv[PT_OUT_FILE]);
        ret = -1;
        goto error;
    }

    if (strncmp(argv[PT_ENC_DIR], "enc", 3) == 0)
        enc = 1;

    e = openssl_engine_init();
    if (e == NULL) {
        ret = -1;
        goto error;
    }
    e_cipher = ENGINE_get_cipher(e, NID_aes_128_cbc);
    if (e_cipher == NULL) {
        ret = -1;
        goto error;
    }

    EVP_CipherInit(&ctx, e_cipher, g_key, g_iv, enc);
    for (;;) {
        inl = fread(g_inbuf, 1, SS_TEST_BUF_SIZE, in);
        if (inl <= 0) {
            if (inl < 0)
                DBG("read(%d) return %d. \n", SS_TEST_BUF_SIZE, inl);
            break;
        }

        if (inl > 0) {
            EVP_CipherUpdate(&ctx, g_outbuf, &outl, g_inbuf, inl);
            DBG("Update: inl %d, outl %d \n", inl, outl);
            fwrite(g_outbuf, 1, outl, out);
        }
    }
    EVP_CipherFinal(&ctx, g_outbuf, &outl);
```

```
    DBG("Update: outl %d \n", outl);
    if (outl > 0)
        fwrite(g_outbuf, 1, outl, out);

error:
    if (in != NULL)
        fclose(in);
    if (out != NULL)
        fclose(out);

    EVP_CIPHER_CTX_cleanup(&ctx);
    openssl_engine_free(e);
    return ret;
}
```

4.3.4 HMAC-SHA1 demo

详细的 demo 文件请查看 openssl-1.0.0/ss_test/目录下。

```
struct af_alg_digest_data
{
    char key[SHA_CBLOCK];
    int keylen;
};

static struct test_st {
    char key[128];
    int key_len;
    char data[128];
    int data_len;
    unsigned char *digest;
} test[] = {
    {   "",
        0,
        "More text test vectors to stuff up EBCDIC machines :-)",
        54,
        (unsigned char *)"b760e92d6662d351eb3801057695ac0346295356",
    },{   "Jefe",
        4,
        "what do ya want for nothing?",
        28,
        (unsigned char *)"effcdf6ae5eb2fa2d27416d5f184df9c259a7c79",
    },{
        {0xaa,0xaa,0xaa,0xaa,0xaa,0xaa,0xaa,0xaa,
         0xaa,0xaa,0xaa,0xaa,0xaa,0xaa,0xaa,0xaa,},
        16,
        {0xdd,0xdd,0xdd,0xdd,0xdd,0xdd,0xdd,0xdd,
         0xdd,0xdd,0xdd,0xdd,0xdd,0xdd,0xdd,0xdd,
         0xdd,0xdd,0xdd,0xdd,0xdd,0xdd,0xdd,0xdd,
         0xdd,0xdd,0xdd,0xdd,0xdd,0xdd,0xdd,0xdd,
         0xdd,0xdd,0xdd,0xdd,0xdd,0xdd,0xdd,0xdd,
         0xdd,0xdd,0xdd,0xdd,0xdd,0xdd,0xdd,0xdd,
         0xdd,0xdd},
        50,
        (unsigned char *)"d730594d167e35d5956fd8003d0db3d3f46dc7bb",
    },{
        {0xaa,0xaa,0xaa,0xaa,0xaa,0xaa,0xaa,0xaa,
```

```
    0xaa,0xaa,0xaa,0xaa,0xaa,0xaa,0xaa,0xaa,
    0xaa,0xaa,0xaa,0xaa},
   20,
   {0xdd,0xdd,0xdd,0xdd,0xdd,0xdd,0xdd,0xdd,
    0xdd,0xdd,0xdd,0xdd,0xdd,0xdd,0xdd,0xdd,
    0xdd,0xdd,0xdd,0xdd,0xdd,0xdd,0xdd,0xdd,
    0xdd,0xdd,0xdd,0xdd,0xdd,0xdd,0xdd,0xdd,
    0xdd,0xdd,0xdd,0xdd,0xdd,0xdd,0xdd,0xdd,
    0xdd,0xdd,0xdd,0xdd,0xdd,0xdd,0xdd,0xdd,
    0xdd,0xdd,0xdd,0xdd,0xdd,0xdd,0xdd,0xdd},
   56,
   (unsigned char *)"09a13335188749ec35ce0dd46185eb6c65719cf2",
  }, {
   {0x01,0x0b,0x0b,0x0b,0x0b,0x0b,0x0b,0x0b,0x0b,
    0x02,0x0b,0x0b,0x0b,0x0b,0x0b,0x0b,0x0b,0x0b,
    0x03,0x0b,0x0b,0x0b,0x0b,0x0b,0x0b,0x0b,0x0b,
    0x04,0x0b,0x0b,0x0b,0x0b,0x0b,0x0b,0x0b,0x0b,
    0x05,0x0b,0x0b,0x0b,0x0b,0x0b,0x0b,0x0b,0x0b,
    0x06,0x0b,0x0b,0x0b,0x0b,0x0b,0x0b,0x0b,0x0b,
    0x07,0x0b,0x0b,0x0b,0x0b,0x0b,0x0b,0x0b,0x0b,
    0x08,0x0b,0x0b,0x0b,0x0b,0x0b,0x0b,0x0b,0x0b,
    0x09},
   65,
   {0xd1,0xdd,0xdd,0xdd,0xdd,0xdd,0xdd,0xdd,
    0xd2,0xdd,0xdd,0xdd,0xdd,0xdd,0xdd,0xdd,
    0xd3,0xdd,0xdd,0xdd,0xdd,0xdd,0xdd,0xdd,
    0xd4,0xdd,0xdd,0xdd,0xdd,0xdd,0xdd,0xdd,
    0xd5,0xdd,0xdd,0xdd,0xdd,0xdd,0xdd,0xdd,
    0xd6,0xdd,0xdd,0xdd,0xdd,0xdd,0xdd,0xdd,
    0xd7,0xdd,0xdd,0xdd,0xdd,0xdd,0xdd,0xdd,
    0xd8,0xdd,0xdd,0xdd,0xdd,0xdd,0xdd,0xdd,
    0xd9,0xdd,0xdd,0xdd,0xdd,0xdd,0xdd,0xdd,
    0xda,0xdd,0xdd,0xdd,0xdd,0xdd,0xdd,0xdd},
   80,
   (unsigned char *)"5422e0af0382e0384f2500f0527d92b7bd3d67c8",
  },
};

static unsigned char md[SHA_DIGEST_LENGTH];

static char *pt(unsigned char *md)
{
  int i;
  static char buf[80] = {0};

  for (i=0; i<SHA_DIGEST_LENGTH; i++)
    sprintf(&(buf[i*2]),"%02x",md[i]);
  return(buf);
}

int check_key(char *dst, char *src, int len)
{
  memset(dst, 0, SHA_CBLOCK);
  if (len <= SHA_CBLOCK) {
    memcpy(dst, src, len);
    return len;
  }

  /* Get the hash value of src. */
  EVP_Digest(src, len, (unsigned char *)dst, NULL, EVP_sha1(), NULL);
```

```
    return SHA_DIGEST_LENGTH;
}

int main(int argc, char *argv[])
{
    int ret = 0;
    unsigned int i = 0;
    char *p = NULL;

    ENGINE *e = NULL;
    EVP_MD_CTX ctx = {0};
    const EVP_MD *e_md = NULL;
    struct af_alg_digest_data *ddata = NULL;

    if (argc == 2)
        i = atoi(argv[1]);
    if (i > 4)
        i = 4;

    e = openssl_engine_init("hmac-sha1");
    if (e == NULL) {
        ret = -1;
        goto error;
    }

    e_md = ENGINE_get_digest(e, NID_hmac_sha1);
    if (e_md == NULL) {
        DBG("ENGINE_get_digest() failed! \n");
        ret = -1;
        goto error;
    }

    EVP_DigestInit(&ctx, e_md);

    ddata = (struct af_alg_digest_data *)ctx.md_data;
    ddata->keylen = check_key(ddata->key, test[i].key, test[i].key_len);

    EVP_DigestUpdate(&ctx, test[i].data, (unsigned long)test[i].data_len);
    EVP_DigestFinal(&ctx, md, NULL);

    p = pt(md);
    if (strcmp(p, (char *)test[i].digest) != 0) {
        printf("HMAC-SHA1 test %d failed!\n", i);
        printf("\tActual: %s \n\tExpect: %s\n", p, test[i].digest);
        ret = 1;
    }
    else
        printf("HMAC-SHA1 test %d ok\n",i);

    EVP_MD_CTX_cleanup(&ctx);
error:
    return ret;
}
```

4.3.5 DH demo

详细的 demo 文件请查看 openssl-1.0.0/ss_test/ss_test/dh.c。

```
void rand_seed_update(void)
{
    static int pos = 0;
    char rnd_seed[] = "string to make the random number generator think it has entropy";

    RAND_seed(&rnd_seed[pos], sizeof rnd_seed);
    pos += 8;
    if (pos >= strlen(rnd_seed))
        pos = 0;
}

int main(int argc, char *argv[])
{
    DH *a;
    DH *b=NULL;
    char buf[12];
    unsigned char *abuf=NULL,*bbuf=NULL;
    int i,alen,blen,aout,bout,ret=-1;
    BIO *out = NULL;
    BIO *in = NULL;

    ENGINE *e = NULL;

    if (argc != 2) {
        printf("You should input as follow: \n");
        printf("\t %s [param file]\n", argv[0]);
        return -1;
    }

    e = openssl_engine_init();
    if (e == NULL)
        goto err;

    CRYPTO_malloc_debug_init();
    CRYPTO_dbg_set_options(V_CRYPTO_MDEBUG_ALL);
    CRYPTO_mem_ctrl(CRYPTO_MEM_CHECK_ON);

    out=BIO_new(BIO_s_file());
    if (out == NULL) EXIT(1);
    BIO_set_fp(out,stdout,BIO_NOCLOSE);

    /* Load DH parameters from a given file. */

    in = BIO_new(BIO_s_file());
    if (BIO_read_filename(in, argv[1]) <= 0)    {
        printf("Failed to open %s \n", argv[1]);
        goto err;
    }

    a = PEM_read_bio_DHparams(in, NULL, NULL, NULL);
    if (a == NULL)    {
        printf("unable to load DH parameters\n");
        goto err;
    }

    BIO_puts(out,"\np: \n");
    BN_print(out,a->p);
    BIO_puts(out,"\ng: \n");
    BN_print(out,a->g);
    BIO_puts(out,"\n\n");
```

```
b = DH_new();
if (b == NULL) goto err;

b->p = BN_dup(a->p);
b->g = BN_dup(a->g);
if ((b->p == NULL) || (b->g == NULL)) goto err;

/* Set a to run with normal modexp and b to use constant time */
a->flags &= ~DH_FLAG_NO_EXP_CONSTTIME;
b->flags |= DH_FLAG_NO_EXP_CONSTTIME;

/* 1.1 a->pub_key = (g ^ a->pri_key) mod p */
rand_seed_update();
if (!DH_generate_key(a)) goto err;
BIO_puts(out,"pri 1: \n");
BN_print(out,a->priv_key);
BIO_puts(out,"\npub 1: \n");
BN_print(out,a->pub_key);
BIO_puts(out,"\n");

/* 1.2 b->pub_key = (g ^ b->pri_key) mod p */
rand_seed_update();
if (!DH_generate_key(b)) goto err;
BIO_puts(out,"pri 2: \n");
BN_print(out,b->priv_key);
BIO_puts(out,"\npub 2: \n");
BN_print(out,b->pub_key);
BIO_puts(out,"\n");

/* 2.1 key1 = (b->pub_key ^ a->pri_key) mod p */
alen=DH_size(a);
abuf=(unsigned char *)OPENSSL_malloc(alen);
aout=DH_compute_key(abuf,b->pub_key,a);

BIO_puts(out,"key1 : \n");
for (i=0; i<aout; i++)
  {
  sprintf(buf,"%02X",abuf[i]);
  BIO_puts(out,buf);
  }
BIO_puts(out,"\n");

/* 2.2 key2 = (a->pub_key ^ b->pri_key) mod p */
blen=DH_size(b);
bbuf=(unsigned char *)OPENSSL_malloc(blen);
bout=DH_compute_key(bbuf,a->pub_key,b);

BIO_puts(out,"key2 : \n");
for (i=0; i<bout; i++)
  {
  sprintf(buf,"%02X",bbuf[i]);
  BIO_puts(out,buf);
  }
BIO_puts(out,"\n\n");

/* Compare key1 and key2 */
if ((aout < 4) || (bout != aout) || (memcmp(abuf,bbuf,aout) != 0))
  {
  fprintf(stderr,"Error in DH routines\n");
```

```
    ret=1;
    }
  else
    ret=0;

  DBG("key1 len = %d, key2 len = %d. [%s]\n", alen, blen, ret==1?"fail" : "OK");

err:
  ERR_print_errors_fp(stderr);

  if (abuf != NULL) OPENSSL_free(abuf);
  if (bbuf != NULL) OPENSSL_free(bbuf);
  if (b != NULL) DH_free(b);
  if (a != NULL) DH_free(a);
  if (in != NULL) BIO_free(in);
  if (out != NULL) BIO_free(out);
#ifdef OPENSSL_SYS_NETWARE
  if (ret) printf("ERROR: %d\n", ret);
#endif

  openssl_engine_free(e);
  EXIT(ret);
  return(ret);
}
```

注意：目前 CE 模块中对称算法、hash 算法以及 rng 算法已注册到 crypto 框架，非对称算法待支持。

5 CE 设备节点方式使用说明

以下是 CE 设备节点调用方式各算法的 demo 用例说明。

5.1 AES demo

重点关注函数 **aes_test**(void) 中的参数 **symm_method**、**aes_mode**、**key_size**、**bit_width** ，按需赋值。

```
/* 通过ioctl传递给内核的数据结构 */
#define AES_DIR_ENCRYPT    0  /* encrypt */
#define AES_DIR_DECRYPT    1  /* decrypt */
/* define the ctx for aes requtest */
typedef struct {
   u32 bit_width;  /* the bitwidth of CFB/CTR mode */
   u32 method;    /* symm method, such as AES/DES/3DES ... */
   u8 *src_buffer;
   u32 src_length;
   u8 *dst_buffer;
   u32 dst_length;
   u8 *key_buffer;
   u32 key_length;
   u8 *iv_buffer;
   u32 iv_length;
   u32 aes_mode;   /* symm mode, such as ECB/CBC/CTR ... */
   u32 dir;        /* encrypt or decrypt */
   u32 ion_flag;
   unsigned long src_phy;
   unsigned long dst_phy;
   unsigned long iv_phy;
   unsigned long key_phy;
   s32 channel_id;
} crypto_aes_req_ctx_t;

/* ioctl cmd: REQUEST -> AES_CRYPTO -> FREE */
#define CE_IOC_MAGIC       'C'
#define CE_IOC_REQUEST     _IOR(CE_IOC_MAGIC, 0, int)  /* requtest */
#define CE_IOC_FREE        _IOW(CE_IOC_MAGIC, 1, int)  /* free */
#define CE_IOC_AES_CRYPTO   _IOW(CE_IOC_MAGIC, 2, crypto_aes_req_ctx_t) /* aes computation */

/* symm mode */
#define AES_MODE_ECB      0
#define AES_MODE_CBC      1
#define AES_MODE_CTR      2
#define AES_MODE_CTS      3
#define AES_MODE_OFB      4
#define AES_MODE_CFB      5
#define AES_MODE_CBC_MAC   6
#define AES_MODE_OCB      7
```

```
#define AES_MODE_GCM      8
#define AES_MODE_XTS      9

/* symm method */
#define SS_METHOD_AES     0x0
#define SS_METHOD_DES     0x1
#define SS_METHOD_3DES    0x2

#define ARRAY_SIZE(a)     ( sizeof(a) / sizeof(a[0]) )
#define sunxi_assert(cond)    { if (!(cond)) { pr_err("ASSERT '%s' failed\n", #cond); exit(-1); } }

static char *symm_mode2str(u32 aes_mode);
static char *symm_method2str(u32 method);
static void __test_AES_enc_dec_cmp(u32 symm_method, u32 aes_mode, u32 data_size, u32 key_size, u32 bit_width,
        char *hardkey);
static void __do_AES_enc_dec_cmp(crypto_aes_req_ctx_t *aes_ctx);

static void aes_test(void)
{
   pr_debug("BEGIN\n");
   /* different data size */
   /* u32 sizes[] = { 16, 256, DATA_SIZE}; */
   u32 sizes[] = { 256 };
   u32 key_size, aes_mode, symm_method;
   u32 bit_width = 0;

   symm_method = SS_METHOD_AES;
   aes_mode = AES_MODE_ECB;
   key_size = 128; /* key_size in 128/192/256 */

   if (aes_mode == AES_MODE_CFB || aes_mode == AES_MODE_CTR)
      /*
       * bit_width in CFB1/CFB8/CFB64/CFB128
       * bit_width in CTR16/CTR32/CTR64/CTR128
       */
      bit_width = 64; /* CFB64 or CTR64 */

   for (int i = 0; i < ARRAY_SIZE(sizes); i++)
      __test_AES_enc_dec_cmp(symm_method, aes_mode, sizes[i], key_size, bit_width);

   pr_debug("SYMM method test success\n\n");
}

static void __test_AES_enc_dec_cmp(u32 symm_method, u32 aes_mode, u32 data_size, u32 key_size, u32 bit_width)
{
   crypto_aes_req_ctx_t *aes_ctx = NULL;
   int i;

   /*
    * key_size is in 128 / 8, 192 / 8, 256 / 8
    * if you measure a different key_size, just change the key array
    */
   u8 key[AES_MAX_KEY_SIZE] = {
      0xFF, 0xEE, 0xDD, 0xCC, 0xBB, 0xAA, 0x99, 0x88,
      0x77, 0x66, 0x55, 0x44, 0x33, 0x22, 0x11, 0x00
   };
   u8 iv[AES_IV_LENGTH] = {
      0x00, 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77,
      0x88, 0x99, 0xAA, 0xBB, 0xCC, 0xDD, 0xEE, 0xFF
   };
```

```
    /* ECB mode has no iv */
    if (aes_mode == AES_MODE_ECB)
        memset(iv, 0x0, sizeof(iv));

    aes_ctx = malloc(sizeof(*aes_ctx));
    sunxi_assert(aes_ctx);
    memset(aes_ctx, 0x0, sizeof(*aes_ctx));

    aes_ctx->src_buffer = malloc(data_size);
    sunxi_assert(aes_ctx->src_buffer);
    aes_ctx->src_length = data_size;

    memset(aes_ctx->src_buffer, 0x88, aes_ctx->src_length);

    aes_ctx->key_length = key_size / 8;

    aes_ctx->key_buffer = malloc(aes_ctx->key_length);
    sunxi_assert(aes_ctx->key_buffer);
    for (i = 0; i < key_size / 8; i++)
        aes_ctx->key_buffer[i] = key[i];

    aes_ctx->iv_buffer = iv;
    aes_ctx->iv_length = sizeof(iv);
    aes_ctx->aes_mode = aes_mode;
    aes_ctx->ion_flag = 0;

    /* only CFB/CTR need set bit_width */
    aes_ctx->bit_width = bit_width;
    aes_ctx->method = symm_method;

    __do_AES_enc_dec_cmp(aes_ctx);

    free(aes_ctx->key_buffer);
    free(aes_ctx->src_buffer);
    free(aes_ctx);
}

static void __do_AES_enc_dec_cmp(crypto_aes_req_ctx_t *aes_ctx)
{
    int err = 0;
    int err1 = 0;
    s32 channel_id = -1;
    u8 *srcbak_buffer = NULL;
    u32 srcbak_length = 0;
    u8 *tmp_buffer = NULL;
    u32 tmp_length = 0;

    pr_debug("BEGIN: symm_method=%s, symm_mode=%s, src_length=%u\n",
        symm_method2str(aes_ctx->method), symm_mode2str(aes_ctx->aes_mode), aes_ctx->src_length);

    if (aes_ctx->src_length & 0xF)  /* non-16-byte alignment */
        aes_ctx->dst_length = ((aes_ctx->src_length + 15) / 16) * 16;  /* dst_length leaves an extra 16 bytes for the unaligned
            part */
    else
        aes_ctx->dst_length = aes_ctx->src_length;
    aes_ctx->dst_buffer = malloc(aes_ctx->dst_length);
    sunxi_assert(aes_ctx->dst_buffer);

    /* get ce fd */
```

```
    int fd = open("/dev/ce", O_RDWR);
    sunxi_assert(fd > 0);

    err = ioctl(fd, CE_IOC_REQUEST, &channel_id);
    sunxi_assert(!err);
    aes_ctx->channel_id = channel_id;

    memset(aes_ctx->dst_buffer, 0x0, aes_ctx->dst_length);
    aes_ctx->dir = AES_DIR_ENCRYPT;
    err = ioctl(fd, CE_IOC_AES_CRYPTO, (unsigned long)aes_ctx);
    if (err) {
        pr_err("ioctl('CE_IOC_AES_CRYPTO') failed\n");
        goto out;
    }

    srcbak_length = aes_ctx->src_length;
    srcbak_buffer = malloc(srcbak_length);
    sunxi_assert(srcbak_buffer);
    memcpy(srcbak_buffer, aes_ctx->src_buffer, aes_ctx->src_length);

    tmp_buffer = aes_ctx->src_buffer;
    aes_ctx->src_buffer = aes_ctx->dst_buffer;
    aes_ctx->dst_buffer = tmp_buffer;
    tmp_length = aes_ctx->src_length;
    aes_ctx->src_length = aes_ctx->dst_length;
    aes_ctx->dst_length = tmp_length;

    memset(aes_ctx->dst_buffer, 0x0, aes_ctx->dst_length);
    aes_ctx->dir = AES_DIR_DECRYPT;
    err = ioctl(fd, CE_IOC_AES_CRYPTO, (unsigned long)aes_ctx);
    if (err) {
        pr_err("ioctl('CE_IOC_AES_CRYPTO') failed\n");
        goto out;
    }

    if (memcmp(srcbak_buffer, aes_ctx->dst_buffer, aes_ctx->dst_length)) {
        pr_err("Origin data:\n");
        sunxi_hexdump(srcbak_buffer, aes_ctx->dst_length);
        pr_err("Encoded-Decoded data:\n");
        sunxi_hexdump(aes_ctx->dst_buffer, aes_ctx->dst_length);
        err = -1;
        goto out;
    }

    err = 0;
    pr_debug("PASS: symm_method=%s, symm_mode=%s\n",
        symm_method2str(aes_ctx->method), symm_mode2str(aes_ctx->aes_mode));

out:
    if (srcbak_buffer)
        free(srcbak_buffer);

    err1 = ioctl(fd, CE_IOC_FREE, &channel_id);
    sunxi_assert(!err1);
    close(fd);

    if (aes_ctx->dst_buffer)
        free(aes_ctx->dst_buffer);

    sunxi_assert(!err);
```

```
}

static char *symm_method2str(u32 method)
{
    char *str;
    switch(method) {
    case SS_METHOD_AES:
        str = "SYMM_METHOD_AES";
        break;
    case SS_METHOD_DES:
        str = "SYMM_METHOD_DES";
        break;
    case SS_METHOD_3DES:
        str = "SYMM_METHOD_3DES";
        break;
    default:
        str = "UNKNOWN_SYMM_METHOD";
        break;
    }

    return str;
}

static char *symm_mode2str(u32 aes_mode)
{
    char *str;
    switch(aes_mode) {
    case AES_MODE_ECB:
        str = "SYMM_MODE_ECB";
        break;
    case AES_MODE_CBC:
        str = "SYMM_MODE_CBC";
        break;
    case AES_MODE_CTR:
        str = "SYMM_MODE_CTR";
        break;
    case AES_MODE_CTS:
        str = "SYMM_MODE_CTS";
        break;
    case AES_MODE_OFB:
        str = "SYMM_MODE_OFB";
        break;
    case AES_MODE_CFB:
        str = "SYMM_MODE_CFB";
        break;
    case AES_MODE_CBC_MAC:
        str = "SYMM_MODE_CBC_MAC";
        break;
    case AES_MODE_OCB:
        str = "SYMM_MODE_OCB";
        break;
    case AES_MODE_GCM:
        str = "SYMM_MODE_GCM";
        break;
    case AES_MODE_XTS:
        str = "SYMM_MODE_XTS";
        break;
    default:
        str = "UNKNOWN_SYMM_MODE";
        break;
```

```
    }
    return str;
}
```

5.2 DES/3DES demo

详见 AES demo，将对应算法的参数赋值为 SS_METHOD_DES/SS_METHOD_3DES 即使用 DES/3DES 算法。

```
/* DES */
symm_method = SS_METHOD_DES;

/* 3DES */
symm_method = SS_METHOD_3DES;
```

5.3 HASH demo

重点关注函数 **hash_test**(void) 中的参数 **hash_mode** ，按需赋值。

```
/* 通过ioctl传递给内核的数据结构 */
/* define the ctx for hash requtest */
typedef struct {
    u8 *text_buffer;
    u32 text_length;
    u8 *key_buffer;    /* hmac */
    u32 key_length;
    u8 *dst_buffer;
    u32 dst_length;
    u8 *iv_buffer;
    u32 iv_length;
    u32 hash_mode;     /* hash mode, such as MD5/SHA1/SHA256 ... */
    u32 ion_flag;
    unsigned long text_phy;
    unsigned long dst_phy;
    s32 channel_id;
} crypto_hash_req_ctx_t;

/* ioctl cmd: REQUEST -> HASH_CRYPTO -> FREE */
#define CE_IOC_MAGIC      'C'
#define CE_IOC_REQUEST    _IOR(CE_IOC_MAGIC, 0, int)  /* request */
#define CE_IOC_FREE       _IOW(CE_IOC_MAGIC, 1, int)  /* free */
#define CE_IOC_HASH_CRYPTO  _IOW(CE_IOC_MAGIC, 4, crypto_hash_req_ctx_t)   /* hash computation */

/* HASH method */
#define SS_METHOD_MD5     0
#define SS_METHOD_SHA1    1
#define SS_METHOD_SHA256  3

#define sunxi_assert(cond)    { if (!(cond)) { pr_err("ASSERT '%s' failed\n", #cond); exit(-1); } }

void hash_digest_test(u32 hash_mode);
```

```
void hash_test(void)
{
    u32 hash_mode;

    /* MD5 */
    hash_mode = SS_METHOD_MD5;
    hash_digest_test(hash_mode);

    /* SHA1 */
    hash_mode = SS_METHOD_SHA1;
    hash_digest_test(hash_mode);

    /* SHA256 */
    hash_mode = SS_METHOD_SHA256;
    hash_digest_test(hash_mode);
}

void hash_digest_test(u32 hash_mode)
{
    u8 *dst_data, *text_data;
    u8 dst_data_size;
    int text_data_size;
    crypto_hash_req_ctx_t *hash_ctx;
    int fd;
    /* config block_size from ce spec */
    int ret, ret1, block_size = 64;
    int channel_id;
    u8 *message_digest;
    u8 digest_md5[16] = { 0xE1, 0xE1, 0xD3, 0xD4, 0x05, 0x73, 0x12, 0x7E,
            0x9E, 0xE0, 0x48, 0x0c, 0xAF, 0x12, 0x83, 0xD6 };
    u8 digest_sha1[20] = { 0x06, 0x57, 0x65, 0x56, 0xD1, 0xAD, 0x80, 0x2F,
            0x24, 0x7C, 0xAD, 0x11, 0xAE, 0x74, 0x8B, 0xE4,
            0x7B, 0x70, 0xCD, 0x9C };
    u8 digest_sha256[32] = { 0x8c, 0x25, 0x74, 0x89, 0x20, 0x63, 0xf9, 0x95,
            0xfd, 0xf7, 0x56, 0xbc, 0xe0, 0x7f, 0x46, 0xc1,
            0xa5, 0x19, 0x3e, 0x54, 0xcd, 0x52, 0x83, 0x7e,
            0xd9, 0x1e, 0x32, 0x00, 0x8c, 0xcf, 0x41, 0xac };

    /* get ce fd */
    fd = open("/dev/ce", O_RDWR);
    sunxi_assert(fd > 0);

    hash_ctx = malloc(sizeof(crypto_hash_req_ctx_t));
    sunxi_assert(fd > 0);

    memset(hash_ctx, 0x0, sizeof(crypto_hash_req_ctx_t));
    hash_ctx->hash_mode = hash_mode;

    /* different hash algorithms have different digest lengths */
    switch (hash_mode) {
    case SS_METHOD_MD5:
        dst_data_size = 16;
        message_digest = digest_md5;
        break;
    case SS_METHOD_SHA1:
        dst_data_size = 20;
        message_digest = digest_sha1;
        break;
    case SS_METHOD_SHA256:
        dst_data_size = 32;
```

```
    message_digest = digest_sha256;
    break;
  default:
    pr_err("unknown hash mode\n");
    ret = -1;
    sunxi_assert(!ret);
  }

  dst_data = malloc(dst_data_size);
  sunxi_assert(dst_data);

  memset(dst_data, 0x00, dst_data_size);
  hash_ctx->dst_buffer = dst_data;
  hash_ctx->dst_length = dst_data_size;

  text_data_size = 1;
  hash_ctx->ion_flag = 0;

  text_data = malloc(text_data_size);
  sunxi_assert(text_data);

  memset(text_data, 0x52, text_data_size);
  hash_ctx->text_buffer = text_data;
  hash_ctx->text_length = text_data_size;

  ret = ioctl(fd, CE_IOC_REQUEST, &channel_id);
  sunxi_assert(!ret);

  hash_ctx->channel_id = channel_id;
  pr_info("channel_id = %d\n", hash_ctx->channel_id);

  /* calculate message digest */
  ret = ioctl(fd, CE_IOC_HASH_CRYPTO, (unsigned long)hash_ctx);
  if (ret) {
    printf("%s: calculate message digest fail\n", __func__);
    goto out;
  }

  printf("\n");
  sunxi_hexdump(hash_ctx->dst_buffer, hash_ctx->dst_length);
  printf("\n");

  if (memcmp(hash_ctx->dst_buffer, message_digest, hash_ctx->dst_length)) {
    pr_err("HASH verify fail\n");
    ret = -1;
    goto out;
  }

  ret = 0;

  pr_debug("HASH verify success\n");
out:
  ret1 = ioctl(fd, CE_IOC_FREE, &channel_id);
  sunxi_assert(!ret1);

  free(hash_ctx->dst_buffer);
  free(hash_ctx->text_buffer);
  free(hash_ctx);
  close(fd);
  sunxi_assert(!ret);
```

```
}
```

5.4 PRNG demo

重点关注函数 **rng_test**(void) 中的参数 **trng** ，按需赋值。

```
/* 通过ioctl传递给内核的数据结构 */
/* define the ctx for rng requtest */
typedef struct {
    u8 *src_buffer;
    u32 src_length;

    u8 *dst_buffer;
    u32 dst_length;

    u8 *key_buffer;
    u32 key_length;

    u8 *iv_buffer;
    u32 iv_length;

    u32 trng;
    u32 reload_offset;

    unsigned long src_phy;
    unsigned long dst_phy;
    unsigned long iv_phy;
    unsigned long key_phy;
    s32 channel_id;
} crypto_rng_req_ctx_t;

/* ioctl cmd: RNG_CRYPTO */
#define CE_IOC_MAGIC          'C'
#define CE_IOC_RNG_CRYPTO   _IOW(CE_IOC_MAGIC, 5, crypto_rng_req_ctx_t) /* rng computation */

#define sunxi_assert(cond)     { if (!(cond)) { pr_err("ASSERT '%s' failed\n", #cond); exit(-1); } }

static void prng_get_random(crypto_rng_req_ctx_t *rng_ctx);

static void rng_test(void)
{
    u32 dst_len, trng;

    dst_len = 20;
    trng = 0;   /* 0:prng 1:trng */

    crypto_rng_req_ctx_t *rng_ctx = NULL;
    u8 key[5] = {
        0x0, 0x1, 0x2, 0x3, 0x4
    };

    rng_ctx = malloc(sizeof(*rng_ctx));
    sunxi_assert(rng_ctx);
    memset(rng_ctx, 0x0, sizeof(*rng_ctx));

    rng_ctx->key_buffer = key;
    rng_ctx->key_length = sizeof(key);
```

```
    rng_ctx->trng = trng;
    rng_ctx->dst_length = dst_len;

    if (trng)
        //trng_get_random(rng_ctx);
        ;
    else
        prng_get_random(rng_ctx);

    free(rng_ctx);
}

static void prng_get_random(crypto_rng_req_ctx_t *rng_ctx)
{
    int err, err1;
    int i, j;
    s32 channel_id;
    u8 pre[20] = {0};
    u8 sec_key[5] = {
        0x5, 0x1, 0x2, 0x3, 0x4
    };

    pr_debug("BEGIN: get random\n");

    rng_ctx->dst_buffer = malloc(rng_ctx->dst_length);
    sunxi_assert(rng_ctx->dst_buffer);

    /* get ce fd */
    int fd = open("/dev/ce", O_RDWR);
    sunxi_assert(fd > 0);

    err = ioctl(fd, CE_IOC_REQUEST, &channel_id);
    sunxi_assert(!err);
    rng_ctx->channel_id = channel_id;

    /* generate random numbers once and save the result to rng_ctx->dst_buffer */
    memset(rng_ctx->dst_buffer, 0x0, rng_ctx->dst_length);
    err = ioctl(fd, CE_IOC_RNG_CRYPTO, (unsigned long)rng_ctx);
    if (err) {
        pr_err("ioctl('CE_IOC_RNG_CRYPTO') failed\n");
        goto out;
    }
    /* save the first generated random number */
    for(i = 0; i < 20; i++)
        pre[i] = rng_ctx->dst_buffer[i];

    rng_ctx->key_buffer = sec_key;
    /* generate random numbers once and save the result to rng_ctx->dst_buffer */
    memset(rng_ctx->dst_buffer, 0x0, rng_ctx->dst_length);
    err = ioctl(fd, CE_IOC_RNG_CRYPTO, (unsigned long)rng_ctx);
    if (err) {
        pr_err("ioctl('CE_IOC_RNG_CRYPTO') failed\n");
        goto out;
    }

    /*
     * compared with the last random number,
     * the random number generated by different seeds should be different.
     *
     * if the data of 5 words is the same as last time,
```

```
    * indicating that the random effect is not good and the result is fail.
    */
    j = 0;
    for(i = 0; i < 20; i++) {
        if(pre[i] == rng_ctx->dst_buffer[i]) {
            j++;
            if(j > 5)
                err = -1;
        }
    }

    if (err) {
        pr_err("Raw data:\n");
        sunxi_hexdump(pre, 20);
        pr_err("Encoded-Decoded data:\n");
        sunxi_hexdump(rng_ctx->dst_buffer, 20);
        goto out;
    }

    err = 0;
    pr_debug("PRNG test success\n");

out:
    err1 = ioctl(fd, CE_IOC_FREE, &channel_id);
    sunxi_assert(!err1);

    close(fd);

    if (rng_ctx->dst_buffer)
        free(rng_ctx->dst_buffer);

    sunxi_assert(!err);
}
```

5.5 CRC demo

```
/* 通过ioctl传递给内核的数据结构 */
/* define the ctx for crc requtest */
typedef struct {
    u8 *src_buffer;
    u32 src_length;
    u32 *dst_buffer;
    u32 dst_length;
    u32 dir;
    u32 width;
    u32 poly;
    u32 init;
    u32 refin;
    u32 refout;
    u32 xorout;
} crypto_crc_req_ctx_t;

/* ioctl cmd: REQUEST -> CRC_CRYPTO -> FREE */
#define CE_IOC_MAGIC            'C'
#define CE_IOC_REQUEST          _IOR(CE_IOC_MAGIC, 0, int)  /* requtest */
#define CE_IOC_FREE             _IOW(CE_IOC_MAGIC, 1, int)  /* free */
```

```
#define CE_IOC_CRC_CRYPTO      _IOW(CE_IOC_MAGIC, 7, crypto_crc_req_ctx_t) /* crc computation */

#define sunxi_assert(cond)     { if (!(cond)) { pr_err("ASSERT '%s' failed\n", #cond); exit(-1); } }

struct crc_mode {
    u32 width;
    u32 poly;
    u32 init;
    u32 refin;
    u32 refout;
    u32 xorout;
    u32 dir;
};

struct crc_item {
    char* str;
    u32 crc;
};

static void crc_test(void)
{
    crypto_crc_req_ctx_t *crc_ctx = NULL;
    int err, i;
    int err1 = 0;
    s32 channel_id = -1;
    static const struct crc_mode crc_test[] = {
        /* width, poly, init, refin, refout, xorout */
        {0, 0x1021, 0, 1, 1, 0},  /* CRC16_CCITT */
        {1, 0x04c11db7, 1, 1, 1, 1}  /* CRC32 */
    };

    static const struct crc_item crc32_test_item[] = {
        {"a", 0xE8B7BE43},
        {"ab", 0x9E83486D},
        {"abc", 0x352441C2}
    };

    pr_debug("BEGIN\n");

    crc_ctx = malloc(sizeof(*crc_ctx));
    sunxi_assert(crc_ctx);
    memset(crc_ctx, 0x0, sizeof(*crc_ctx));

    crc_ctx->dst_length = sizeof(u32);
    crc_ctx->dst_buffer = malloc(crc_ctx->dst_length);
    sunxi_assert(crc_ctx->dst_buffer);
    memset(crc_ctx->dst_buffer, 0x0, crc_ctx->dst_length);

    /* get ce fd */
    int fd = open("/dev/ce", O_RDWR);
    sunxi_assert(fd > 0);

    pr_debug("\n\nSTART CRC32 TEST:\n");

    crc_ctx->dir = crc_test[1].dir;
    crc_ctx->width = crc_test[1].width;
    crc_ctx->poly = crc_test[1].poly;
    crc_ctx->init = crc_test[1].init;
    crc_ctx->refin = crc_test[1].refin;
    crc_ctx->refout = crc_test[1].refout;
```

```
    crc_ctx->xorout = crc_test[1].xorout;

    for (i = 0; i < sizeof(crc32_test_item)/sizeof(crc32_test_item[0]); i++) {
        crc_ctx->src_length = strlen((void*)crc32_test_item[i].str);
        crc_ctx->src_buffer = malloc(crc_ctx->src_length);
        sunxi_assert(crc_ctx->src_buffer);
        memcpy(crc_ctx->src_buffer, crc32_test_item[i].str, crc_ctx->src_length);

        pr_debug("CRC32_TEXT_%d: expected val = 0x%x\n", i, crc32_test_item[i].crc);
        sleep(1);

        err = ioctl(fd, CE_IOC_CRC_CRYPTO, (unsigned long)crc_ctx);
        if (err) {
            pr_err("ioctl('CE_IOC_CRC_CRYPTO') failed\n");
            free(crc_ctx->src_buffer);
            goto out;
        }

        pr_debug("actual val = 0x%x\n\n", *crc_ctx->dst_buffer);

        if (*crc_ctx->dst_buffer != crc32_test_item[i].crc) {
            pr_debug("crc32(%s) faild: %04x != %04x\n", crc32_test_item[i].str, *crc_ctx->dst_buffer, crc32_test_item[i].crc);
            free(crc_ctx->src_buffer);
            goto out;
        }

        free(crc_ctx->src_buffer);
    }

out:
    free(crc_ctx->dst_buffer);
    free(crc_ctx);
}
```

⚠ 注意

CE crc 算法的 ioctl 模式仅 V1 版本支持。

6 Linux CRYPTO API 使用说明

因为 CE 的接口已经注册到内核的 crypto 的框架之中，因此如果需要在内核态中调用 CE 的接口，只需要调用内核 crypto 的接口即可。

6.1 hash 接口

由于算法太多，这里就不一一列举了，这里以 hash 算法为例，首先查看 include/crypto/hash.h 。这里定义每个 hash 接口定义，而且还有相关的描述:

```
/**
 * crypto_ahash_init() - (re)initialize message digest handle
 * @req: ahash_request handle that already is initialized with all necessary
 *    data using the ahash_request_* API functions
 *
 * The call (re-)initializes the message digest referenced by the ahash_request
 * handle. Any potentially existing state created by previous operations is
 * discarded.
 *
 * Return: 0 if the message digest initialization was successful; < 0 if an
 *      error occurred
 */
static inline int crypto_ahash_init(struct ahash_request *req)
{
   struct crypto_ahash *tfm = crypto_ahash_reqtfm(req);

   if (crypto_ahash_get_flags(tfm) & CRYPTO_TFM_NEED_KEY)
      return -ENOKEY;

   return tfm->init(req);
}

/**
 * crypto_ahash_update() - add data to message digest for processing
 * @req: ahash_request handle that was previously initialized with the
 *    crypto_ahash_init call.
 *
 * Updates the message digest state of the &ahash_request handle. The input data
 * is pointed to by the scatter/gather list registered in the &ahash_request
 * handle
 *
 * Return: 0 if the message digest update was successful; < 0 if an error
 *      occurred
 */
static inline int crypto_ahash_update(struct ahash_request *req)
{
   return crypto_ahash_reqtfm(req)->update(req);
}
```

著作权声明

本文档及内容受著作权法保护，其著作权由珠海全志科技股份有限公司（“全志”）拥有并保留一切权利。

本文档是全志的原创作品和版权财产，未经全志书面许可，任何单位和个人不得擅自摘抄、复制、修改、发表或传播本文档内容的部分或全部，且不得以任何形式传播。

商标声明


、 
、 
、 
（不完全列举）均为珠海全志科技股份有限公司的商标或者注册商标。在本文档描述的产品中出现的其它商标，产品名称，和服务名称，均由其各自所有人拥有。

免责声明

您购买的产品、服务或特性应受您与珠海全志科技股份有限公司（“全志”）之间签署的商业合同和条款的约束。本文档中描述的全部或部分产品、服务或特性可能不在您所购买或使用的范围内。使用前请认真阅读合同条款和相关说明，并严格遵循本文档的使用说明。您将自行承担任何不当使用行为（包括但不限于如超压，超频，超温使用）造成的不利后果，全志概不负责。

本文档作为使用指导仅供参考。由于产品版本升级或其他原因，本文档内容有可能修改，如有变更，恕不另行通知。全志尽全力在本文档中提供准确的信息，但并不确保内容完全没有错误，因使用本文档而发生损害（包括但不限于间接的、偶然的、特殊的损失）或发生侵犯第三方权利事件，全志概不负责。本文档中的所有陈述、信息和建议并不构成任何明示或暗示的保证或承诺。

本文档未以明示或暗示或其他方式授予全志的任何专利或知识产权。在您实施方案或使用产品的过程中，可能需要获得第三方的权利许可。请您自行向第三方权利人获取相关的许可。全志不承担也不代为支付任何关于获取第三方许可的许可费或版税（专利税）。全志不对您所使用的第三方许可技术做出任何保证、赔偿或承担其他义务。