



Linux CE 开发指南

版本号: 2.0
发布日期: 2021.04.02

版本历史

| 版本号 | 日期 | 制/修订人 | 内容描述 |
|-----|------------|---------|-----------------|
| 1.0 | 2020.7.19 | AWA0480 | 添加基础模板 |
| 2.0 | 2021.04.02 | AWA0480 | 适配 linux-5.4 平台 |



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 | 1 |
| 1.4 相关术语 | 1 |
| 2 CE 模块描述 | 3 |
| 2.1 CE 的算法支持 | 3 |
| 2.2 CE 框架设计 | 3 |
| 2.3 模块配置介绍 | 4 |
| 2.3.1 Device Tree 配置说明 | 4 |
| 2.3.2 CE 的 kernel menuconfig 的配置说明 | 5 |
| 2.3.2.1 linux-4.9 内核的配置 | 5 |
| 2.3.2.2 linux-5.4 内核的配置 | 7 |
| 2.3.2.3 网络的配置 | 8 |
| 2.3.3 采用 ARM 的加速指令的配置 | 9 |
| 2.4 源码结构介绍 | 10 |
| 2.4.1 linux-4.9 源代码结构 | 10 |
| 2.4.2 linux-5.4 源代码结构 | 12 |
| 3 模块接口描述 | 14 |
| 3.1 算法注册接口 | 14 |
| 3.1.1 crypto_register_alg() | 14 |
| 3.1.1.1 函数原型 | 14 |
| 3.1.1.2 功能描述 | 14 |
| 3.1.1.3 返回值 | 14 |
| 3.1.1.4 参数说明 | 14 |
| 3.1.2 crypto_unregister_alg() | 15 |
| 3.1.2.1 函数原型 | 15 |
| 3.1.2.2 功能描述 | 15 |
| 3.1.2.3 返回值 | 15 |
| 3.1.2.4 参数说明 | 15 |
| 3.1.3 crypto_register_ahash() | 16 |
| 3.1.3.1 函数原型 | 16 |
| 3.1.3.2 功能描述 | 16 |
| 3.1.3.3 返回值 | 16 |
| 3.1.3.4 参数说明 | 16 |
| 3.1.4 crypto_unregister_ahash() | 17 |
| 3.1.4.1 函数原型 | 17 |
| 3.1.4.2 功能描述 | 17 |
| 3.1.4.3 返回值 | 17 |
| 3.1.4.4 参数说明 | 17 |

| | |
|---------------------------------|-----------|
| 3.2 算法处理接口 | 17 |
| 3.2.1 ss_aes_start() | 18 |
| 3.2.1.1 函数原型 | 18 |
| 3.2.1.2 功能描述 | 18 |
| 3.2.1.3 返回值 | 18 |
| 3.2.1.4 参数说明 | 18 |
| 3.2.2 ss_hash_start | 19 |
| 3.2.2.1 函数原型 | 19 |
| 3.2.2.2 功能描述 | 19 |
| 3.2.2.3 返回值 | 19 |
| 3.2.2.4 参数说明 | 19 |
| 3.2.3 ss_rng_start() | 20 |
| 3.2.3.1 函数原型 | 20 |
| 3.2.3.2 功能描述 | 20 |
| 3.2.3.3 返回值 | 20 |
| 3.2.3.4 参数说明 | 20 |
| 4 openssl 的接口 | 22 |
| 4.1 openssl 的代码库 | 22 |
| 4.2 openssl 的配置与编译 | 22 |
| 4.2.1 openssl 的配置 | 22 |
| 4.2.2 openssl 的编译说明 | 22 |
| 4.2.3 openssl 的库文件的生成 | 23 |
| 4.3 openssl demo 用例说明 | 23 |
| 4.3.1 使用 af_alg 引擎 | 23 |
| 4.3.2 MD5 demo | 24 |
| 4.3.3 AES demo | 25 |
| 4.3.4 HMAC-SHA1 demo | 27 |
| 4.3.5 DH demo | 30 |
| 5 Linux CRYPTO API 使用说明 | 33 |
| 5.1 hash 接口 | 33 |

插 图

| | |
|------------------------------------|----|
| 2-1 Linux CE 体系结构图 | 3 |
| 2-2 Cryptographic API 配置 | 5 |
| 2-3 Cryptographic API 配置 | 6 |
| 2-4 Cryptographic API 配置 | 6 |
| 2-5 Cryptographic API 配置 | 7 |
| 2-6 Cryptographic API 配置 | 7 |
| 2-7 Cryptographic API 配置 | 8 |
| 2-8 Cryptographic API 配置 | 8 |
| 2-9 NET 配置选项 | 9 |
| 2-10 ARM 加速指令的配置 | 10 |
| 2-11 ARM 加速指令的配置 | 10 |



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: 适用产品列表

| 内核版本 | 驱动文件 |
|-----------|---------------------------|
| Linux-4.9 | drivers/crypto/sunxi-ss/* |
| Linux-5.4 | drivers/crypto/sunxi-ce/* |

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

- API: Application Program Interface 应用程序接口
- SUNXI: 指 Allwinner 的一系列 SOC 硬件平台
- 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
- CE: Crypto Engine, 算法引擎，以前称作 SS
- AES: Advanced Encryption Standard, 高级加密标准
- CRC32: Cyclic redundancy check 32, 循环冗余校验 (32 位)
- DES: Data Encryption Standard, 数据加密标准
- 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
- ECB: Electronic Code Book mode, 电子密码本模式
- CBC: Cipher Block Chaining mode, 加密块链模式
- CFB: Cipher feedback, 密码反馈模式

- CTR: Counter mode, 计数模式
- CTS: Ciphertext Stealing mode
- OFB: Output feedback, 输出反馈模式
- XTS: XEX-based tweaked-codebook mode with ciphertext stealing
- DH: Diffie-Hellman 算法, 密码一致协议
- 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
- MD5: Message Digest Algorithm 5, 消息摘要算法第五版
- SHA: Secure Hash Algorithm, 安全散列算法
- 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
- IDMA: Internal DMA
- RSA: 公钥加密算法
- TRNG: True Random Number Generator, 真随机数发生器
- PRNG: Pseudo Random Number Generator, 伪随机数发生器



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 CE 框架设计

CE 驱动按照 Linux 内核中的 crypto 框架设计，在应用层能够 and 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：

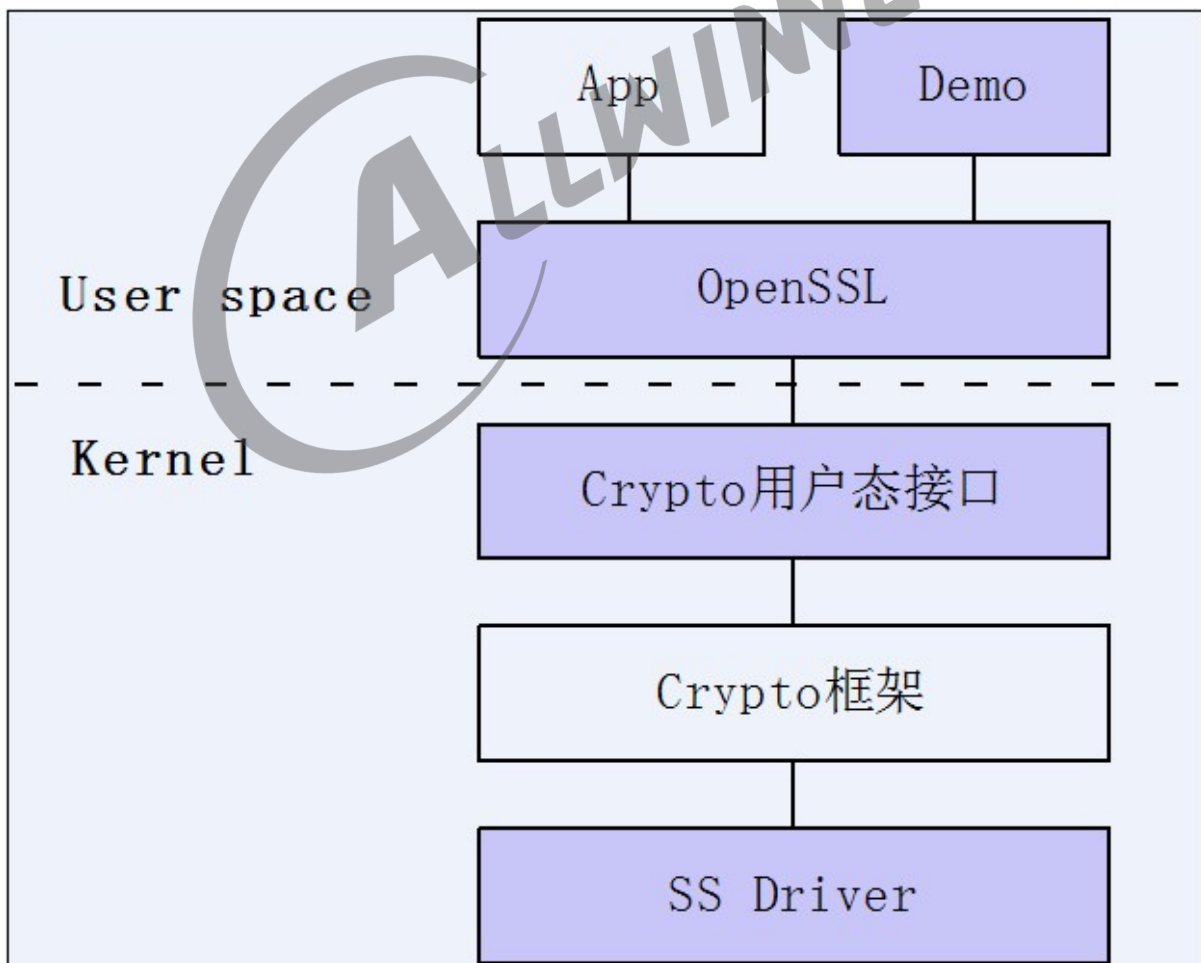


图 2-1: Linux CE 体系结构图

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）。

2.3 模块配置介绍

2.3.1 Device Tree 配置说明

在 Device Tree 中对 CE 控制器进行配置，如下：

```
cryptoengine: ce@03040000 {
    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_MBUS_CE>,
        <&ccu CLK_PLL_PERIPH0_2X>;
    clock-names = "bus_ce", "ce_clk", "mbus_ce", "pll_periph0_2x";
    resets = <&ccu RST_BUS_CE>;
    status = "okay";
};
```

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。

2.3.2 CE 的 kernel menuconfig 的配置说明

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

2.3.2.1 linux-4.9 内核的配置

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

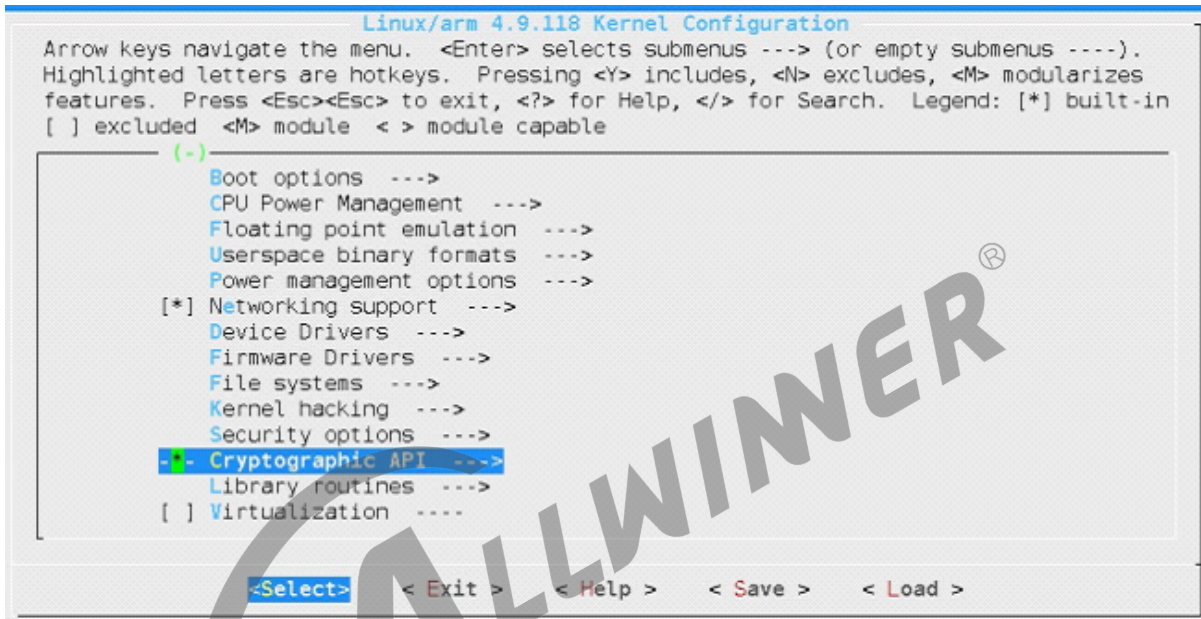


图 2-2: Cryptographic API 配置

2. 然后，选中 Disable run-time selftests 选项，进入下一级配置，如图所示：

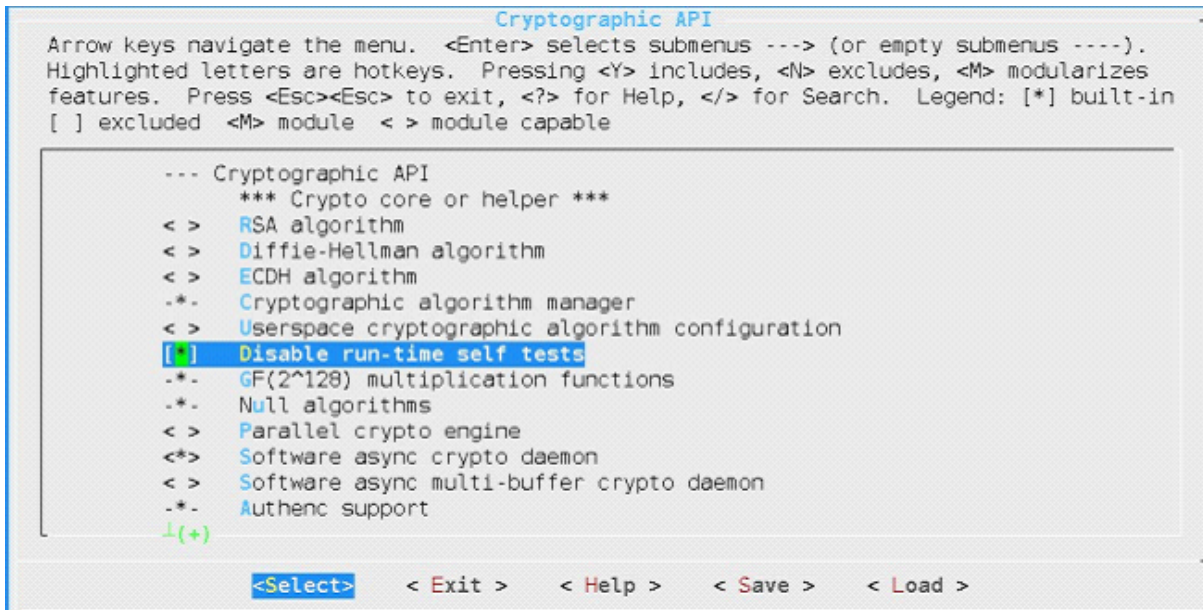


图 2-3: Cryptographic API 配置

3. 接着增加 crypto 框架的用户态接口支持, 选中下图的四项, 如图所示:

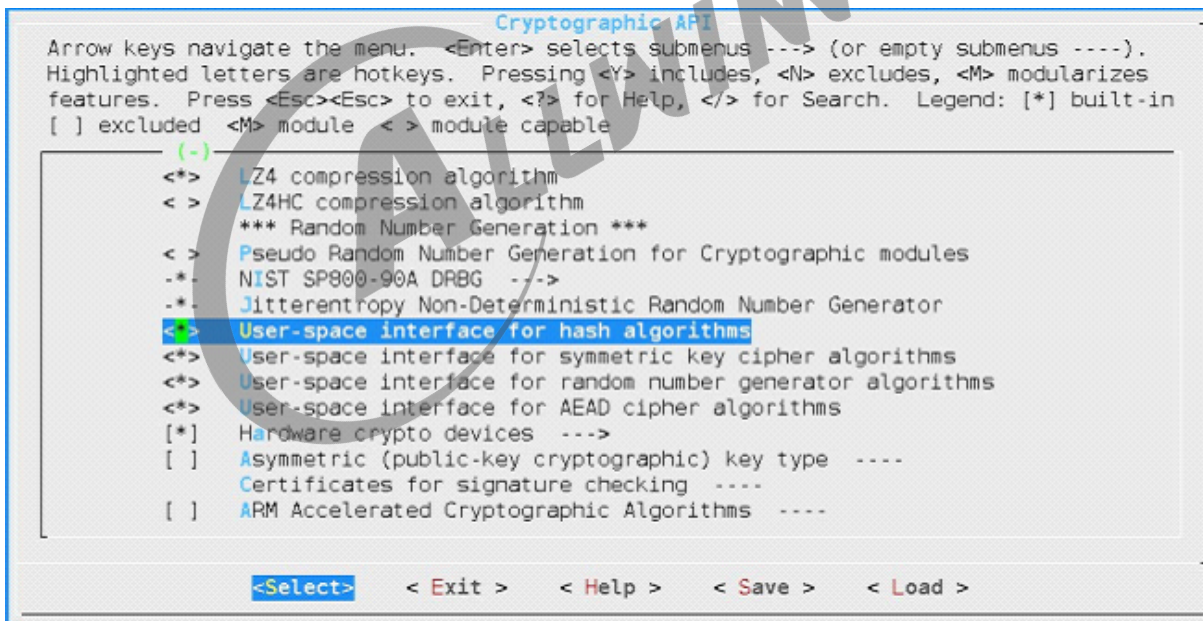


图 2-4: Cryptographic API 配置

4. 在 Hardware crypto devies 中选择 CE 驱动, 如图所示:

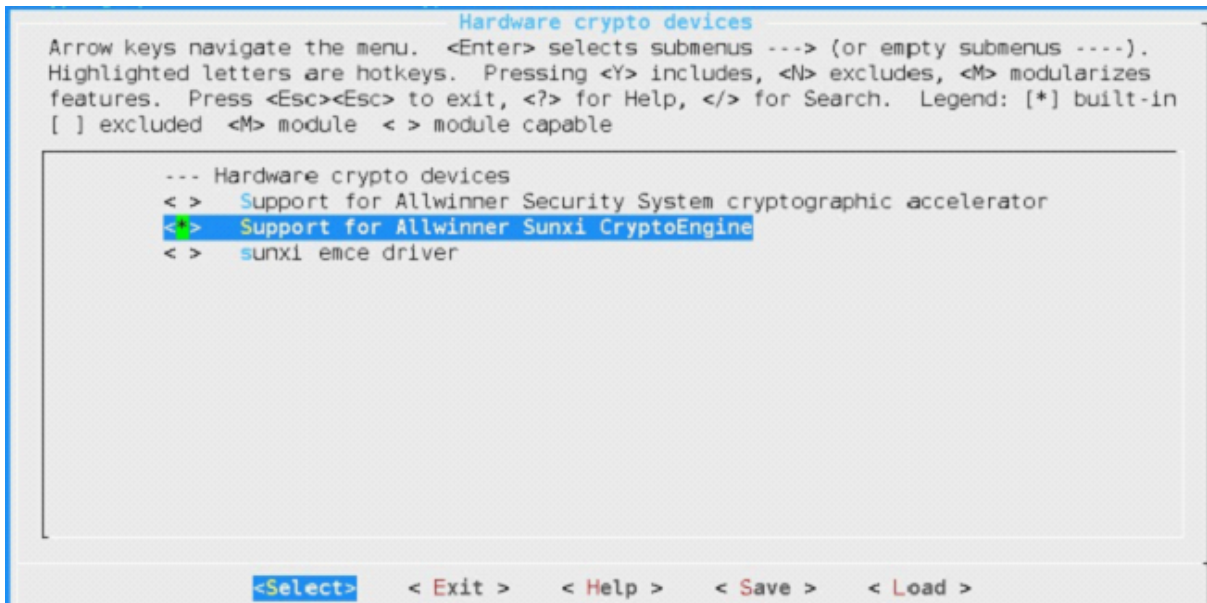


图 2-5: Cryptographic API 配置

2.3.2.2 linux-5.4 内核的配置

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

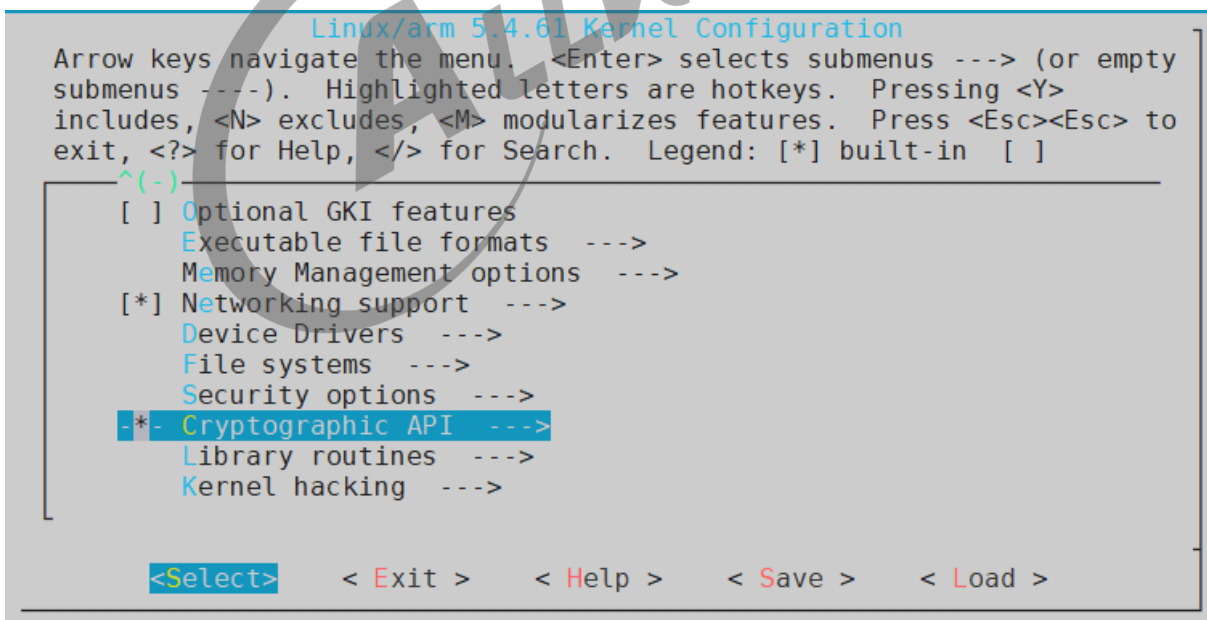


图 2-6: Cryptographic API 配置

2. 接着增加 crypto 框架的用户态接口支持, 选中下图的四项, 如图所示:

```

Cryptographic API
Arrow keys navigate the menu. <Enter> selects submenus ---> (or empty
submenus ----). Highlighted letters are hotkeys. Pressing <Y>
includes, <N> excludes, <M> modularizes features. Press <Esc><Esc> to
exit, <?> for Help, </> for Search. Legend: [*] built-in [ ]
^(-)
< > Zstd compression algorithm
    *** Random Number Generation ***
< > Pseudo Random Number Generation for Cryptographic modules
< > NIST SP800-90A DRBG ----
< > Jitterentropy Non-Deterministic Random Number Generator
[*] User-space interface for hash algorithms
[*] User-space interface for symmetric key cipher algorithms
[*] User-space interface for random number generator algorithms
[*] User-space interface for AEAD cipher algorithms
[*] Hardware crypto devices --->
↓(+)
```

<Select> < Exit > < Help > < Save > < Load >

图 2-7: Cryptographic API 配置

3. 在 Hardware crypto devies 中选择 CE 驱动, 如图所示:

```

Hardware crypto devies
Arrow keys navigate the menu. <Enter> selects submenus ---> (or empty
submenus ----). Highlighted letters are hotkeys. Pressing <Y>
includes, <N> excludes, <M> modularizes features. Press <Esc><Esc> to
exit, <?> for Help, </> for Search. Legend: [*] built-in [ ]
--- Hardware crypto devies
< > Support for Microchip / Atmel ECC hw accelerator
< > Support for Microchip / Atmel SHA accelerator and RNG
< > Support for Allwinner Security System cryptographic acceler
[*] Support for Allwinner Sunxi CryptoEngine
< > Inside Secure's SafeXcel cryptographic engine driver
< > Support for ARM TrustZone CryptoCell family of security pro
```

<Select> < Exit > < Help > < Save > < Load >

图 2-8: Cryptographic API 配置

2.3.2.3 网络的配置

如果采用 openssl 进行调用 CE 的 API 接口进行加解密, 需要保证 CONFIG_NET 是打开的。如图所示:

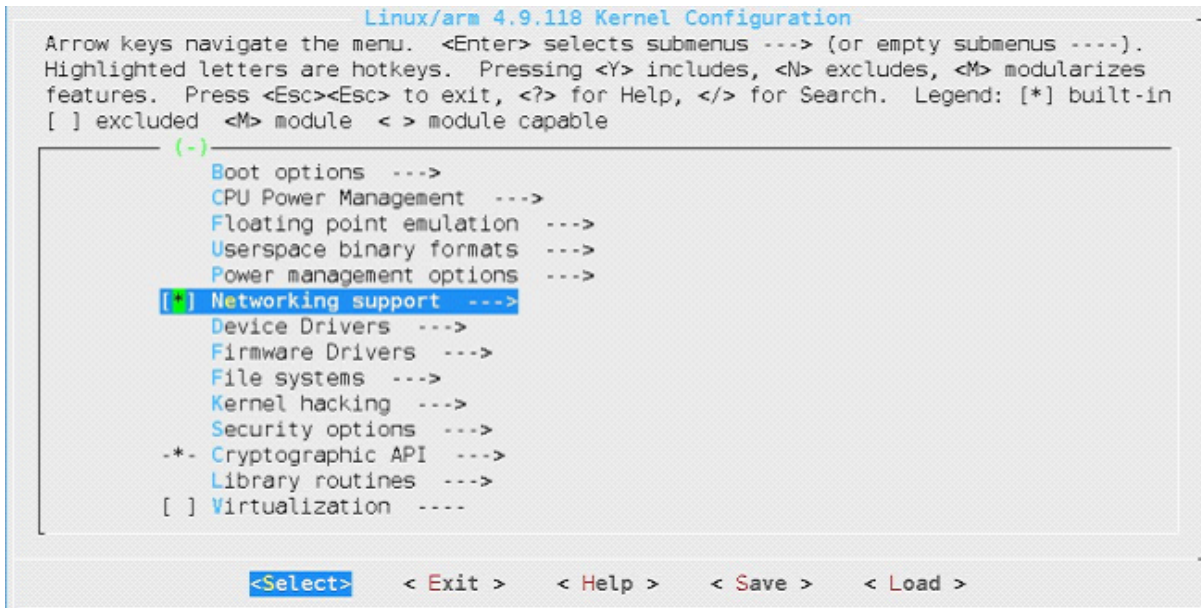


图 2-9: NET 配置选项

2.3.3 采用 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-V7 架构的加速指令的进行配置:

1. 首先, 选择 ARM Accelerated Cryptographic Algorithm 选项进入下一级配置, 如图所示:

```
Linux/arm 5.4.61 Kernel Configuration
Arrow keys navigate the menu. <Enter> selects submenus ---> (or empty
submenus ----). Highlighted letters are hotkeys. Pressing <Y>
includes, <N> excludes, <M> modularizes features. Press <Esc><Esc> to
exit, <?> for Help, </> for Search. Legend: [*] built-in [ ]
^(-)
System Type --->
Bus support --->
Kernel Features --->
Boot options --->
CPU Power Management --->
Floating point emulation --->
Power management options --->
Firmware Drivers --->
[*] ARM Accelerated Cryptographic Algorithms --->
[ ] Virtualization ----
↑(+)
<Select> < Exit > < Help > < Save > < Load >
```

图 2-10: ARM 加速指令的配置

2. 首先, 选择进行配置相应的算法, 如图所示:

```
ARM Accelerated Cryptographic Algorithms
Arrow keys navigate the menu. <Enter> selects submenus ---> (or empty
submenus ----). Highlighted letters are hotkeys. Pressing <Y>
includes, <N> excludes, <M> modularizes features. Press <Esc><Esc> to
exit, <?> for Help, </> for Search. Legend: [*] built-in [ ]
--- ARM Accelerated Cryptographic Algorithms
< > SHA1 digest algorithm (ARM-asm) (NEW)
[*] SHA-224/256 digest algorithm (ARM-asm and NEON)
< > SHA-384/512 digest algorithm (ARM-asm and NEON) (NEW)
< > Scalar AES cipher for ARM (NEW)
<Select> < Exit > < Help > < Save > < Load >
```

图 2-11: ARM 加速指令的配置

2.4 源码结构介绍

2.4.1 linux-4.9 源代码结构

CE 驱动的源代码位于内核在 `drivers/crypto/sunxi-ss` 目录下:

```
drivers/crypto/sunxi-ss/  
├─ sunxi_ss.c // Sunxi平台的SS算法注册、处理流程的实现  
├─ sunxi_ss.h // 为Sunxi平台的SS驱动定义了一些共用的宏、数据结构  
├─ sunxi_ss_proc.h // 各种算法的接口声明  
├─ sunxi_ss_proc_comm.h // 三个版本SS控制器共用的算法处理过程  
├─ V1  
│   ├── sunxi_ss_proc.c // V1版本SS控制器的算法处理过程  
│   ├── sunxi_ss_reg.c // V1版本SS控制器的寄存器接口实现  
│   └─ sunxi_ss_reg.h // V1版本SS控制器的寄存器接口声明  
├─ V2  
│   ├── sunxi_ss_proc.c // V2版本SS控制器的算法处理过程  
│   ├── sunxi_ss_reg.c // V2版本SS控制器的寄存器接口实现  
│   └─ sunxi_ss_reg.h // V2版本SS控制器的寄存器接口声明  
├─ V3  
│   ├── sunxi_ss_proc.c // V3版本SS控制器的算法处理过程  
│   ├── sunxi_ss_reg.c // V3版本SS控制器的寄存器接口实现  
│   └─ sunxi_ss_reg.h // V3版本SS控制器的寄存器接口声明  
├─ V4  
│   ├── sunxi_ss_proc.c // V4版本SS控制器的算法处理过程  
│   ├── sunxi_ss_reg.c // V4版本SS控制器的寄存器接口实现  
│   └─ sunxi_ss_reg.h // V4版本SS控制器的寄存器接口声明
```

通过 Makefile 控制四种 CE 版本的源码编译，linux-4.9/drivers/crypto/sunxi-ss/Makefile 内容如下

```
obj-$(CONFIG_CRYPTO_DEV_SUN4I_SS) += sun4i-ss.o  
sun4i-ss-y += sun4i/sun4i-ss-core.o sun4i/sun4i-ss-hash.o sun4i/sun4i-ss-cipher.o  
  
obj-$(CONFIG_CRYPTO_DEV_SUNXI) += ss.o  
  
ss-y += sunxi_ss.o sunxi_ss_proc_comm.o sunxi_ce_cdev_comm.o  
#ss-y += ss_kernel_test.o  
  
ifdef CONFIG_ARCH_SUN8IW11  
    SUNXI_SS_VER = v3  
endif  
ifdef CONFIG_ARCH_SUN8IW12  
    SUNXI_SS_VER = v3  
endif  
ifdef CONFIG_ARCH_SUN8IW15  
    SUNXI_SS_VER = v3  
endif  
ifdef CONFIG_ARCH_SUN8IW17  
    SUNXI_SS_VER = v3  
endif  
ifdef CONFIG_ARCH_SUN8IW7  
    SUNXI_SS_VER = v3  
endif  
ifdef CONFIG_ARCH_SUN8IW18  
    SUNXI_SS_VER = v3  
endif  
ifdef CONFIG_ARCH_SUN50I  
    SUNXI_SS_VER = v3  
endif  
ifdef CONFIG_ARCH_SUN8IW16  
    SUNXI_SS_VER = v3  
endif  
ifdef CONFIG_ARCH_SUN8IW19
```



```

    SUNXI_SS_VER = v3
endif
ifdef CONFIG_ARCH_SUN50IW8
    SUNXI_SS_VER = v4
endif
ifdef CONFIG_ARCH_SUN50IW10
    SUNXI_SS_VER = v4
endif
ss-y += $(SUNXI_SS_VER)/sunxi_ss_reg.o $(SUNXI_SS_VER)/sunxi_ss_proc.o

ccflags-y += -Idrivers/crypto/sunxi-ss/$(SUNXI_SS_VER)

#ccflags-y += -DDEBUG

```

2.4.2 linux-5.4 源代码结构

CE 驱动的源代码位于内核在 drivers/crypto/sunxi-ce 目录下:

```

drivers/crypto/sunxi-ce/
├── drivers
│   ├── crypto
│   │   ├── sunxi-ce
│   │   │   ├── Kconfig
│   │   │   ├── Makefile
│   │   │   ├── sun4i
│   │   │   │   ├── sun4i-ss-cipher.c
│   │   │   │   ├── sun4i-ss-core.c
│   │   │   │   ├── sun4i-ss.h
│   │   │   │   └── sun4i-ss-hash.c
│   │   │   ├── sunxi_ce.c // Sunxi平台的SS算法注册、处理流程的实现
│   │   │   ├── sunxi_ce_cdev_comm.c
│   │   │   ├── sunxi_ce.h // 为Sunxi平台的SS驱动定义了一些共用的宏、数据结构
│   │   │   ├── sunxi_ce_proc_comm.c // 各版本CE控制器共用的算法处理过程
│   │   │   ├── sunxi_ce_proc.h // 各种算法的接口声明
│   │   │   ├── v2
│   │   │   │   ├── sunxi_ce_proc.c // V1版本CE控制器的算法处理过程
│   │   │   │   ├── sunxi_ce_reg.c // V1版本CE控制器的寄存器接口实现
│   │   │   │   └── sunxi_ce_reg.h // V1版本CE控制器的寄存器接口声明
│   │   │   ├── v3
│   │   │   │   ├── sunxi_ce_proc.c
│   │   │   │   ├── sunxi_ce_proc_walk.c
│   │   │   │   ├── sunxi_ce_reg.c
│   │   │   │   └── sunxi_ce_reg.h
│   │   │   ├── v4
│   │   │   │   ├── sunxi_ce_proc.c
│   │   │   │   ├── sunxi_ce_reg.c
│   │   │   │   └── sunxi_ce_reg.h
│   │   │   └── v5
│   │   │       ├── sunxi_ce_proc.c
│   │   │       ├── sunxi_ce_reg.c
│   │   │       └── sunxi_ce_reg.h

```

通过 Makefile 控制四种 CE 版本的源码编译，linux-5.4/drivers/crypto/sunxi-ce/Makefile 内容如下

```
obj-$(CONFIG_CRYPTO_DEV_SUN4I_SS) += sun4i-ss.o
sun4i-ss-y += sun4i/sun4i-ss-core.o sun4i/sun4i-ss-hash.o sun4i/sun4i-ss-cipher.o

obj-$(CONFIG_CRYPTO_DEV_SUNXI) += sunxi-ce.o

sunxi-ce-$(CONFIG_CRYPTO_DEV_SUNXI) += sunxi_ce.o sunxi_ce_proc_comm.o
#ss-y += ss_kernel_test.o

ifdef CONFIG_ARCH_SUN20IW1
    SUNXI_CE_VER = v3
endif
ifdef CONFIG_ARCH_SUN8IW11
    SUNXI_CE_VER = v3
endif
ifdef CONFIG_ARCH_SUN8IW12
    SUNXI_CE_VER = v3
endif
ifdef CONFIG_ARCH_SUN8IW15
    SUNXI_CE_VER = v3
endif
ifdef CONFIG_ARCH_SUN8IW17
    SUNXI_CE_VER = v3
endif
ifdef CONFIG_ARCH_SUN8IW7
    SUNXI_CE_VER = v3
endif
ifdef CONFIG_ARCH_SUN8IW18
    SUNXI_CE_VER = v3
endif
ifdef CONFIG_ARCH_SUN50I
    SUNXI_CE_VER = v3
endif
ifdef CONFIG_ARCH_SUN8IW16
    SUNXI_CE_VER = v3
endif
ifdef CONFIG_ARCH_SUN8IW19
    SUNXI_CE_VER = v3
endif
ifdef CONFIG_ARCH_SUN8IW20
    SUNXI_CE_VER = v3
endif
ifdef CONFIG_ARCH_SUN50IW8
    SUNXI_CE_VER = v4
endif
ifdef CONFIG_ARCH_SUN50IW10
    SUNXI_CE_VER = v4
endif
ifdef CONFIG_ARCH_SUN50IW12
    SUNXI_CE_VER = v5
endif

sunxi-ce-y += $(SUNXI_CE_VER)/sunxi_ce_reg.o $(SUNXI_CE_VER)/sunxi_ce_proc.o

ccflags-y += -I$(srctree)/drivers/crypto/sunxi-ce/$(SUNXI_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, 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;
};
```

3.1.4 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）

```
/* 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;
```

3.2.2 ss_hash_start

3.2.2.1 函数原型

```
static int ss_hash_start(ss_hash_ctx_t *ctx, ss_aes_req_ctx_t *req_ctx, int len)
```

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

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];
    int md_size;
    int cnt; /* in Byte */
} ss_hash_ctx_t;
```

备注: 为了兼容 V3.2 的硬件 Padding 功能, 这个函数增加了一个参数 last, 用来表示是否最后一包。

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, unsigned int dlen)
```

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)

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];  
    int md_size;  
    int cnt; /* in Byte */  
} ss_hash_ctx_t;
```



4 openssl 的接口

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。

4.1 openssl 的代码库

openssl 的代码库都已经上传 Gerrit 上。

```
git clone ssh://yourname@gerrit.allwinnertech.com:29418/longan/platfrom/framework/openssl/openssl-1.0.0
```

4.2 openssl 的配置与编译

如果应用层想采用硬件 CE 进行开发, 则需要利用 openssl 标准的接口, 才能调用 CE 驱动。

4.2.1 openssl 的配置

openssl 现在代码库, 已经适配好一些标准算法和架构平台的配置, 并且和 longan 的配置文件绑定在一起了, 因此只需要在 longan 下进行配置即可。

```
$cd longan
$./build.sh config
```

4.2.2 openssl 的编译说明

openssl 现在代码库, 已经和 longan 的 liunx 的编译工具绑定在一起了, 因此当 longan 配置好后, 进行如下编译

```
$cd openssl-1.0.0
$make clean
$make
```

4.2.3 openssl 的库文件的生成

如果应用层用 openssl 进行开发, 则需要包含 openssl 的库文件进行开发。openssl 的库文件的生成, 命令如下:

```
$cd openssl-1.0.0
$make install
```

执行成功后, 会在 openssl-1.0.0/out 目录下生成以下文件:

```
openssl-1.0.0/out
├─ usr
│   └─ ssl
│       ├── bin      // OpenSSL可执行文件
│       ├── certs   // 目前为空, 可存放数据证书
│       ├── include // OpenSSL的接口头文件
│       ├── lib     // OpenSSL会用到的动态库, cd 包括所有engine
│       ├── man     // 帮助手册 (man命令需要的格式)
│       ├── misc    // 其他工具ls
│       └─ openssl.cnf // OpenSSL的配置文件
└─ private // 目前为空
```

应用层在进行开发时, 需要链接 lib 目录下 3 个动态库文件:

```
├─ libcrypto.so.1.0.0
├─ libssl.so.1.0.0
└─ libaf_alg.so
```

4.3 openssl demo 用例说明

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。

```
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},
      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},
    56,
    (unsigned char *)"09a13335188749ec35ce0dd46185eb6c65719cf2",
}, {
    {0x01, 0x0b, 0x0b, 0x0b, 0x0b, 0x0b, 0x0b, 0x0b,
    0x02, 0x0b, 0x0b, 0x0b, 0x0b, 0x0b, 0x0b, 0x0b,
    0x03, 0x0b, 0x0b, 0x0b, 0x0b, 0x0b, 0x0b, 0x0b,
    0x04, 0x0b, 0x0b, 0x0b, 0x0b, 0x0b, 0x0b, 0x0b,
    0x05, 0x0b, 0x0b, 0x0b, 0x0b, 0x0b, 0x0b, 0x0b,
    0x06, 0x0b, 0x0b, 0x0b, 0x0b, 0x0b, 0x0b, 0x0b,
    0x07, 0x0b, 0x0b, 0x0b, 0x0b, 0x0b, 0x0b, 0x0b,
    0x08, 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->key_len = 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

```
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);
}
```



5 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。

5.1 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

版权所有 © 2021 珠海全志科技股份有限公司。保留一切权利。

本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、发表或传播本档内容的部分或全部，且不得以任何形式传播。

商标声明

、 **全志科技** （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。