AX CIPHER API 文档

文档版本：V1.3

发布日期：2024/02/20

目 录

权利声明

注意

您购买的产品、服务或特性等应受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的购买或使用范围之内。除非商业合同另有约定，本公司对本文档内容不做任何明示或默示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

前 言

本文档将简要 AX620E 中的 AX_CIPHER 模块，提供该模块使用的 API、数据结构以及使用过程中可能会产生的错误码。

适用产品

AX620E 系列产品（AX630C、AX620Q）

适读人群

- 技术支持工程师
- 软件开发工程师

符号与格式定义

符号/格式	说明
`xxx`	表示您可以执行的命令行。
斜体	表示变量。如，“*安装目录*/AX620E_SDK_Vx.x.x/build 目录”中的“安装目录”是一个变量，由您的实际环境决定。
☞ 说明/备注：	表示您在使用产品的过程中，我们向您说明的事项。
！ 注意：	表示您在使用产品的过程中，需要您特别注意的事项。

修订历史

文档版本	发布时间	修订说明
V1.0	2023/07/17	文档初版
V1.1	2023/10/12	添加 crypto 框架支持的描述
V1.2	2024/01/30	添加 rsa 加解密 api 描述
V1.3	2024/02/20	添加 rsa 签名和验签对 sha384 和 sha512 模式的支持

1 概述

1.1 概述

CIPHER 是 AX620E 处理器集成的安全算法处理模块。它可以支持 AES 和 DES/3DES 的对称加解密算法，RSA 不对称加解密算法，随机数生成，以及支持 HASH 摘要算法，主要用于对于音视频的加解密保护。不同算法支持的工作模式如下表所示：

表1-1 不同算法支持的工作模式

算法	支持的工作模式
AES	ECB/CBC/CTR/ICM/XTS/F8
DES/3DES	ECB/CBC
RSA	1024/2048/3072
HASH	SHA1/SHA224/SHA256/SHA384/SHA512
随机数	DRBG/TRNG

1.2 驱动介绍

Cipher 驱动位于 kernel 的 drivers/crypto/axera 目录，驱动支持 linux crypto 框架，并且支持使用 MISC 设备的方式访问硬件；对应的用户态可以使用 kernel 原生的 Netlink AF_ALG 协议访问 cipher 驱动，也可以使用 SDK 提供的基于 MISC 设备方式的 API 接口。

1.3 驱动使能

选择 Cryptographic API —> Hardware crypto devices —> axera cryptographic engine driver 即可将 cipher 的驱动编译进 kernel，也可单独编译成模块，使用时再加载。

```
--- Hardware crypto devices
< >   Support for Microchip / Atmel ECC hw accelerator
< >   Support for Microchip / Atmel SHA accelerator and RNG
[ ]   Support for AMD Secure Processor
< >   VirtIO crypto driver
< >   Inside Secure's SafeXcel cryptographic engine driver
< >   Support for ARM TrustZone CryptoCell family of security processors
< >   Support for Hisilicon SEC crypto block cipher accelerator
< >   Support for amlogic cryptographic offloader
<*>   axera cryptographic engine driver  --->
```

可以进一步进入 axera cryptographic engine driver 选项通过 Support for axera cipher crypto Engine 和 Support for axera cipher misc api 来配置是否支持 kernel crypto 框架，或者是否支持使用 MISC 设备的方式访问硬件。

```
    -*- axera cryptographic engine driver
    [*]   Support for axera cipher crypto Engine
    [*]   Support for axera cipher misc api
```

2 API 参考

AX_CIPHER_Init

【描述】

初始化 CIPHER 模块。

【语法】

```
AX_S32 AX_CIPHER_Init(AX_VOID)
```

【参数】

无

【返回值】

返回值	描述
0	成功
非 0	失败，返回错误码

【需求】

- 头文件：ax_cipher_api.h
- 库文件：libax_cipher.so
- KO 文件：ax_cipher.ko

【注意】

无

【举例】

无

【相关主题】

AX_CIPHER_DeInit()

AX_CIPHER_DeInit

【描述】

关闭 CIPHER 模块。

【语法】

```
AX_S32 AX_CIPHER_DeInit(AX_VOID)
```

【参数】

无

【返回值】

返回值	描述
0	成功
非 0	失败，其值为错误码

【需求】

- 头文件：ax_cipher_api.h
- 库文件：libax_cipher.so
- KO 文件：ax_cipher.ko

【注意】

无

【举例】

无

【相关主题】

无

AX_CIPHER_CreateHandle

【描述】

创建一路 CIPHER 句柄。

【语法】

```
AX_S32 AX_CIPHER_CreateHandle(AX_CIPHER_HANDLE *phCipher, const
AX_CIPHER_CTRL_T *pstCipherCtrl)
```

【参数】

参数名称	描述	输入/输出
phCipher	CIPHER 句柄指针	输入
pstCipherCtrl	CIPHER 属性指针	输入

【返回值】

返回值	描述
0	成功
非 0	失败，其值为错误码

【需求】

- 头文件：ax_cipher_api.h
- 库文件：libax_cipher.so
- KO 文件：ax_cipher.ko

【注意】

AX_CIPHER_CreateHandle 需要在 AX_CIPHER_Init 之后调用。

【举例】

参考 sample_cipher.c。

AX_CIPHER_Encrypt

【描述】

对数据进行 AES、DES 等加密。

【语法】

```
AX_S32 AX_CIPHER_Encrypt(AX_CIPHER_HANDLE pCipher, AX_U8 *szSrcAddr, AX_U8 *szDestAddr, AX_U32 byteLength)
```

【参数】

参数名称	描述	输入/输出
pCipher	CIPHER 的句柄	输入
szSrcAddr	源数据（待加密的数据）的虚拟地址	输入
szDestAddr	目标数据（加密后的数据的）虚拟地址	输入(非空)
byteLength	加密数据的长度	输入

【返回值】

返回值	描述
0	成功
非 0	失败，其值为错误码

【需求】

- 头文件：ax_cipher_api.h
- 库文件：libax_cipher.so
- KO 文件：ax_cipher.ko

【注意】

CIPHER 句柄必须已经创建。

【举例】

参考 sample_aes.c。

AX_CIPHER_Decrypt

【描述】

对数据进行 AES、DES 等解密。

【语法】

```
AX_S32 AX_CIPHER_Decrypt(AX_CIPHER_HANDLE pCipher, AX_U8 *szSrcAddr, AX_U8 *szDestAddr, AX_U32 byteLength)
```

【参数】

参数名称	描述	输入/输出
pCipher	CIPHER 的句柄	输入
szSrcAddr	源数据（待解密的数据）的虚拟地址	输入
szDestAddr	目标数据（解密后的数据的）虚拟地址	输入
byteLength	解密数据的长度	输入

【返回值】

返回值	描述
0	成功
非 0	失败，其值为错误码

【需求】

- 头文件：ax_cipher_api.h
- 库文件：libax_cipher.so
- KO 文件：ax_cipher.ko

【注意】

无

【举例】

参考 sample_aes.c。

AX_CIPHER_DestroyHandle

【描述】

销毁一路 CIPHER。

【语法】

```
AX_S32 AX_CIPHER_DestroyHandle(AX_CIPHER_HANDLE pCipher)
```

【参数】

参数名称	描述	输入/输出
pCipher	CIPHER 的句柄	输入

【返回值】

返回值	描述
0	成功
非 0	失败，其值为错误码

【需求】

- 头文件：ax_cipher_api.h
- 库文件：libax_cipher.so
- KO 文件：ax_cipher.ko

【注意】

创建句柄和销毁句柄要成对使用。

【举例】

参考 sample_cipher.c。

AX_CIPHER_RsaVerify

【描述】

使用 RSA 公钥验证一段文本。

【语法】

```
AX_S32 AX_CIPHER_RsaVerify(AX_CIPHER_RSA_PUBLIC_KEY *key, AX_U8 *msg, AX_U32 msgBytes, AX_CIPHER_SIG_DATA_T *sig)
```

【参数】

参数名称	描述	输入/输出
Key	验证签名的公钥属性信息指针	输入
Msg	待验证的数据	输入
msgBytes	待验证的数据的长度	输入
Sig	待验证的签名数据指针	输入

【返回值】

返回值	描述
0	成功
非 0	失败，其值为错误码

【需求】

- 头文件：ax_cipher_api.h
- 库文件：libax_cipher.so
- KO 文件：ax_cipher.ko

【注意】

无

【举例】

参考 sample_rsa.c。

AX_CIPHER_RsaSign

【描述】

使用 RSA 公钥签名一段文本。

【语法】

```
AX_S32 AX_CIPHER_RsaSign(AX_CIPHER_RSA_PRIVATE_KEY *key, AX_U8 *msg, AX_U32 msgBytes, AX_CIPHER_SIG_DATA_T *sig)
```

【参数】

参数名称	描述	输入/输出
Key	签名的私钥属性信息指针	输入
Msg	待签名的数据	输入
msgBytes	待签名的数据的长度	输入
Sig	签名结果数据指针	输出

【返回值】

返回值	描述
0	成功
非 0	失败，其值为错误码

【需求】

- 头文件：ax_cipher_api.h
- 库文件：libax_cipher.so
- KO 文件：ax_cipher.ko

【注意】

无

【举例】

参考 sample_rsa.c。

AX_CIPHER_RsaPublicEncrypt

【描述】

使用 RSA 公钥加密一段文本。

【语法】

```
AX_U32 AX_CIPHER_RsaPublicEncrypt(AX_CIPHER_RSA_PUB_ENC_T *pRsaEnc, AX_U8
*pInput, AX_U32 inLen, AX_U8 *pOutput, AX_U32 *pOutLen);
```

【参数】

参数名称	描述	输入/输出
pRsaEnc	RSA 加解密的公钥结构体	输入
pInput	待加密的数据	输入
inLen	待加密的数据的长度	输入
pOutput	加密结果	输出
pOutLen	加密结果长度	输出

【返回值】

返回值	描述
0	成功
非 0	失败，其值为错误码

【需求】

- 头文件：ax_cipher_api.h
- 库文件：libax_cipher.so
- KO 文件：ax_cipher.ko

【注意】

无

【举例】

参考 sample_rsa.c。

AX_CIPHER_RsaPublicDecrypt

【描述】

使用 RSA 公钥解密一段密文。

【语法】

```
AX_CIPHER_RsaPublicDecrypt(AX_CIPHER_RSA_PUB_ENC_T *pRsaDec, AX_U8 *pInput,
AX_U32 inLen, AX_U8 *pOutput, AX_U32 *pOutLen);
```

【参数】

参数名称	描述	输入/输出
pRsaEnc	RSA 加解密的公钥结构体	输入
pInput	待解密的数据	输入
inLen	待解密的数据的长度	输入
pOutput	解密结果	输出
pOutLen	解密结果长度	输出

【返回值】

返回值	描述
0	成功
非 0	失败，其值为错误码

【需求】

- 头文件：ax_cipher_api.h
- 库文件：libax_cipher.so
- KO 文件：ax_cipher.ko

【注意】

无

【举例】

参考 sample_rsa.c。

AX_CIPHER_RsaPrivateEncrypt

【描述】

使用 RSA 私钥加密一段文本。

【语法】

```
AX_U32 AX_CIPHER_RsaPrivateEncrypt(AX_CIPHER_RSA_PRI_ENC_T *pRsaEnc, AX_U8
*pInput, AX_U32 inLen, AX_U8 *pOutput, AX_U32 *pOutLen);
```

【参数】

参数名称	描述	输入/输出
pRsaEnc	RSA 加解密的私钥结构体	输入
pInput	待加密的数据	输入
inLen	待加密的数据的长度	输入
pOutput	加密结果	输出
pOutLen	加密结果长度	输出

【返回值】

返回值	描述
0	成功
非 0	失败，其值为错误码

【需求】

- 头文件：ax_cipher_api.h
- 库文件：libax_cipher.so
- KO 文件：ax_cipher.ko

【注意】

无

【举例】

参考 sample_rsa.c。

AX_CIPHER_RsaPrivateDecrypt

【描述】

使用 RSA 私钥解密一段密文。

【语法】

```
AX_U32 AX_CIPHER_RsaPrivateDecrypt(AX_CIPHER_RSA_PRI_ENC_T *pRsaDec, AX_U8
*pInput, AX_U32 inLen, AX_U8 *pOutput, AX_U32 *pOutLen);
```

【参数】

参数名称	描述	输入/输出
pRsaEnc	RSA 加解密的私钥结构体	输入
pInput	待解密的数据	输入
inLen	待解密的数据的长度	输入
pOutput	解密结果	输出
pOutLen	解密结果长度	输出

【返回值】

返回值	描述
0	成功
非 0	失败，其值为错误码

【需求】

- 头文件：ax_cipher_api.h
- 库文件：libax_cipher.so
- KO 文件：ax_cipher.ko

【注意】

无

【举例】

参考 sample_rsa.c。

AX_CIPHER_EncryptPhy

【描述】

使用物理地址对数据进行 AES、DES 等加密。

【语法】

```
AX_S32 AX_CIPHER_EncryptPhy(AX_CIPHER_HANDLE pCipher, AX_U64 szSrcAddr,
AX_U64 szDestAddr, AX_U32 byteLength)
```

【参数】

参数名称	描述	输入/输出
pCipher	CIPHER 的句柄	输入
szSrcAddr	源数据（待加密的数据）的物理地址	输入
szDestAddr	目标数据（加密后的数据的）物理地址	输入
byteLength	加密数据的长度	输入

【返回值】

返回值	描述
0	成功
非 0	失败，其值为错误码

【需求】

- 头文件：ax_cipher_api.h
- 库文件：libax_cipher.so
- KO 文件：ax_cipher.ko

【注意】

CIPHER 句柄必须已经创建。

【举例】

参考 sample_aes.c。

AX_CIPHER_DecryptPhy

【描述】

使用物理地址对数据进行 AES、DES 等解密。

【语法】

```
AX_S32 AX_CIPHER_DecryptPhy(AX_CIPHER_HANDLE pCipher, AX_U64 szSrcAddr,
AX_U64 szDestAddr, AX_U32 byteLength)
```

【参数】

参数名称	描述	输入/输出
pCipher	CIPHER 的句柄	输入
szSrcAddr	源数据（待解密的数据）的物理地址	输入
szDestAddr	目标数据（解密后的数据的）物理地址	输入
byteLength	解密数据的长度	输入

【返回值】

返回值	描述
0	成功
非 0	失败，其值为错误码

【需求】

- 头文件：ax_cipher_api.h
- 库文件：libax_cipher.so
- KO 文件：ax_cipher.ko

【注意】

无

【举例】

参考 sample_aes.c。

AX_CIPHER_HashInit

【描述】

初始化 HASH 模块。

【语法】

```
AX_S32 AX_CIPHER_HashInit(AX_CIPHER_HASH_CTL_T *pstHashCtl, AX_CIPHER_HANDLE *pHashHandle)
```

【参数】

参数名称	描述	输入/输出
pstHashCtl	用于计算 hash 的配置信息	输入
pHashHandle	输出的 hash 的句柄	输出

【返回值】

返回值	描述
0	成功
非 0	失败，其值为错误码

【需求】

- 头文件：ax_cipher_api.h
- 库文件：libax_cipher.so
- KO 文件：ax_cipher.ko

【注意】

无

【举例】

参考 sample_hash.c。

AX_CIPHER_HashUpdate

【描述】

更新 Hash 计算的数据。

【语法】

```
AX_S32 AX_CIPHER_HashUpdate(AX_CIPHER_HANDLE handle, AX_U8 *inputData, AX_U32 inPutLen)
```

【参数】

参数名称	描述	输入/输出
Handle	计算 hash 的句柄	输入
inputData	待计算 hash 的数据	输入
inPutLen	待计算的 hash 数据的长度	输入

【返回值】

返回值	描述
0	成功
非 0	失败，其值为错误码

【需求】

- 头文件：ax_cipher_api.h
- 库文件：libax_cipher.so
- KO 文件：ax_cipher.ko

【注意】

- 输入的数据的长度必须为 64 字节对齐，不能为 0，最后一个 block 无此限制。
- Hash 的句柄必须创建。

【举例】

参考 sample_hash.c。

AX_CIPHER_HashFinal

【描述】

获取 Hash 的值。

【语法】

```
AX_S32 AX_CIPHER_HashFinal(AX_CIPHER_HANDLE handle, AX_U8 *inputData, AX_U32 inPutLen, AX_U8 *outPutHash)
```

【参数】

参数名称	描述	输入/输出
Handle	计算 hash 的句柄	输入
inputData	待计算 hash 的数据	输入
inPutLen	待计算的 hash 数据的长度	输入
outPutHash	输出 hash 值的指针	输出

【返回值】

返回值	描述
0	成功
非 0	失败，其值为错误码

【需求】

- 头文件：ax_cipher_api.h
- 库文件：libax_cipher.so
- KO 文件：ax_cipher.ko

【注意】

- 针对一次 hash 计算，如果已经调用了 AX_CIPHER_HashUpdate，则再次调用 AX_CIPHER_HashFinal 时，数据长度不能为 0。

- 如果直接调用 AX_CIPHER_HashFinal 函数，数据长度可以为 0。
- AX_CIPHER_ALGO_HASH_SHA1 时，hash 值长度为 20
- AX_CIPHER_ALGO_HASH_SHA224 时，hash 值长度为 28
- AX_CIPHER_ALGO_HASH_SHA256 时，hash 值长度为 32
- AX_CIPHER_ALGO_HASH_SHA384 时，hash 值长度为 48
- AX_CIPHER_ALGO_HASH_SHA512 时，hash 值长度为 64
- AX_CIPHER_ALGO_MAC_HMAC_SHA1 时，hash 值长度为 20
- AX_CIPHER_ALGO_MAC_HMAC_SHA224 时，hash 值长度为 28
- AX_CIPHER_ALGO_MAC_HMAC_SHA256 时，hash 值长度为 32
- AX_CIPHER_ALGO_MAC_HMAC_SHA384 时，hash 值长度为 48
- AX_CIPHER_ALGO_MAC_HMAC_SHA512 时，hash 值长度为 64

【举例】

参考 sample_hash.c。

AX_CIPHER_HashUpdatePhy

【描述】

更新 Hash 计算的数据。

【语法】

```
AX_S32 AX_CIPHER_HashUpdatePhy(AX_CIPHER_HANDLE handle, AX_U64 inputPhy,
AX_U32 inPutLen)
```

【参数】

参数名称	描述	输入/输出
Handle	计算 hash 的句柄	输入
inputPhy	待计算 hash 数据的物理地址	输入
inPutLen	待计算的 hash 数据的长度	输入

【返回值】

返回值	描述
0	成功
非 0	失败，其值为错误码

【需求】

- 头文件：ax_cipher_api.h
- 库文件：libax_cipher.so
- KO 文件：ax_cipher.ko

【注意】

- 输入的数据的长度必须为 64 字节对齐，不能为 0，最后一个 block 无此限制。
- Hash 的句柄必须创建。

【举例】

参考 sample_hash.c。

AX_CIPHER_HashFinalPhy

【描述】

获取 Hash 的值。

【语法】

```
AX_S32 AX_CIPHER_HashFinalPhy(AX_CIPHER_HANDLE handle, AX_U64 inputPhy,
AX_U32 inPutLen, AX_U8 *outPutHash)
```

【参数】

参数名称	描述	输入/输出
Handle	计算 hash 的句柄	输入
inputPhy	待计算 hash 数据的物理地址	输入
inPutLen	待计算的 hash 数据的长度	输入
outPutHash	输出 hash 值的指针	输出

【返回值】

返回值	描述
0	成功
非 0	失败，其值为错误码

【需求】

- 头文件：ax_cipher_api.h
- 库文件：libax_cipher.so
- KO 文件：ax_cipher.ko

【注意】

- 针对一次 hash 计算，如果已经调用了 AX_CIPHER_HashUpdate，则再次调用 AX_CIPHER_HashFinal 时，数据长度不能为 0。

- 如果直接调用 AX_CIPHER_HashFinal 函数，数据长度可以为 0。
- AX_CIPHER_ALGO_HASH_SHA1 时，hash 值长度为 20
- AX_CIPHER_ALGO_HASH_SHA224 时，hash 值长度为 28
- AX_CIPHER_ALGO_HASH_SHA256 时，hash 值长度为 32
- AX_CIPHER_ALGO_HASH_SHA384 时，hash 值长度为 48
- AX_CIPHER_ALGO_HASH_SHA512 时，hash 值长度为 64
- AX_CIPHER_ALGO_MAC_HMAC_SHA1 时，hash 值长度为 20
- AX_CIPHER_ALGO_MAC_HMAC_SHA224 时，hash 值长度为 28
- AX_CIPHER_ALGO_MAC_HMAC_SHA256 时，hash 值长度为 32
- AX_CIPHER_ALGO_MAC_HMAC_SHA384 时，hash 值长度为 48
- AX_CIPHER_ALGO_MAC_HMAC_SHA512 时，hash 值长度为 64

【举例】

参考 sample_hash.c。

AX_CIPHER_GetRandomNumber

【描述】

生成随机数。

【语法】

```
AX_U32 AX_CIPHER_GetRandomNumber(AX_U32 *pRandomNumber, AX_U32 size)
```

【参数】

参数名称	描述	输入/输出
pRandomNumber	输出随机数的指针	输出
Size	输出随机数的长度	输入

【返回值】

返回值	描述
0	成功
非 0	失败，其值为错误码

【需求】

- 头文件：ax_cipher_api.h
- 库文件：libax_cipher.so
- KO 文件：ax_cipher.ko

【注意】

数据的长度不能为 0，数据的长度需要小于 65535。

【举例】

参考 sample_trng.c。

3 数据结构

AX_CIPHER_ALGO_E

【说明】

定义 CIPHER 加密算法。

【定义】

```
typedef enum

{

    AX_CIPHER_ALGO_HASH_SHA1   = 0,          // SHA-1

    AX_CIPHER_ALGO_HASH_SHA224 = 1,          // SHA-224

    AX_CIPHER_ALGO_HASH_SHA256 = 2,          // SHA-256

    AX_CIPHER_ALGO_HASH_SHA384 = 3,          // SHA-384

    AX_CIPHER_ALGO_HASH_SHA512 = 4,          // SHA-512

    AX_CIPHER_ALGO_MAC_HMAC_SHA1 = 5,         // HMAC-SHA-1

    AX_CIPHER_ALGO_MAC_HMAC_SHA224 = 6,       // HMAC-SHA-224

    AX_CIPHER_ALGO_MAC_HMAC_SHA256 = 7,       // HMAC-SHA-256

    AX_CIPHER_ALGO_MAC_HMAC_SHA384 = 8,       // HMAC-SHA-384

    AX_CIPHER_ALGO_MAC_HMAC_SHA512 = 9,       // HMAC-SHA-512

    AX_CIPHER_ALGO_MAC_AES_CMAC = 10,         // AES-CMAC

    AX_CIPHER_ALGO_MAC_AES_CBC_MAC = 11,      // AES-CBC-MAC

    AX_CIPHER_ALGO_CIPHER_AES = 12,           // AES
```

```
    AX_CIPHER_ALGO_CIPHER_DES = 13,            // DES

    AX_CIPHER_ALG_INVALID = 0xffffffff,

}AX_CIPHER_ALGO_E;
```

【成员】

成员名称	描述
AX_CIPHER_ALGO_HASH_SHA1	SHA1 哈希算法
AX_CIPHER_ALGO_HASH_SHA224	SHA224 哈希算法
AX_CIPHER_ALGO_HASH_SHA256	SHA256 哈希算法
AX_CIPHER_ALGO_HASH_SHA384	SHA384 哈希算法
AX_CIPHER_ALGO_HASH_SHA512	SHA512 哈希算法
AX_CIPHER_ALGO_MAC_HMAC_SHA1	HMAC_SHA1 哈希算法
AX_CIPHER_ALGO_MAC_HMAC_SHA224	HMAC_SHA224 哈希算法
AX_CIPHER_ALGO_MAC_HMAC_SHA256	HMAC_SHA256 哈希算法
AX_CIPHER_ALGO_MAC_HMAC_SHA384	HMAC_SHA384 哈希算法
AX_CIPHER_ALGO_MAC_HMAC_SHA512	HMAC_SHA512 哈希算法
AX_CIPHER_ALGO_MAC_AES_CMAC	AES_CMAC 算法
AX_CIPHER_ALGO_MAC_AES_CBC_MAC	AES_CBC_MAC 算法
AX_CIPHER_ALGO_CIPHER_AES	AES 算法
AX_CIPHER_ALGO_CIPHER_DES	DES 算法
AX_CIPHER_ALG_INVALID	非法值

【注意事项】

无

【相关数据类型及接口】

无

AX_CIPHER_MODE_E

【说明】

定义 CIPHER 的工作模式。

【定义】

```
typedef enum

{

    // (Block)Cipher modes

    AX_CIPHER_MODE_CIPHER_ECB = 0,       // ECB

    AX_CIPHER_MODE_CIPHER_CBC,          // CBC

    AX_CIPHER_MODE_CIPHER_CTR,          // CTR

    AX_CIPHER_MODE_CIPHER_ICM,          // ICM

    AX_CIPHER_MODE_CIPHER_F8,           // F8

    AX_CIPHER_MODE_CIPHER_CCM,          // CCM

    AX_CIPHER_MODE_CIPHER_XTS,          // XTS

    AX_CIPHER_MODE_CIPHER_GCM,          // GCM

    AX_CIPHER_MODE_CIPHER_MAX,          // must be last

} AX_CIPHER_MODE_E;
```

【成员】

成员名称	描述
AX_CIPHER_MODE_CIPHER_ECB	电码本模式（Electronic Codebook Book (ECB)
AX_CIPHER_MODE_CIPHER_CBC	密码分组链接模式（Cipher Block Chaining (CBC)）
AX_CIPHER_MODE_CIPHER_CTR	计算器模式（Counter (CTR)）
AX_CIPHER_MODE_CIPHER_ICM	ICM 模式

成员名称	描述
AX_CIPHER_MODE_CIPHER_F8	F8 模式
AX_CIPHER_MODE_CIPHER_CCM	CCM（Counter with CBC-MAC）模式
AX_CIPHER_MODE_CIPHER_XTS	XTS 模式
AX_CIPHER_MODE_CIPHER_GCM	GCM(Galois/Counter Mode)模式
AX_CIPHER_MODE_CIPHER_MAX	无效值

【注意事项】

无

【相关数据类型及接口】

无

AX_CIPHER_RSA_SIGN_SCHEME_E

【说明】

定义 RSA 签名的算法。

【定义】

```
typedef enum AX_CIPHER_RSA_SIGN_SCHEME_E

{

    AX_CIPHER_RSA_SIGN_RSASSA_PKCS1_V15_SHA1 = 0x0,

    AX_CIPHER_RSA_SIGN_RSASSA_PKCS1_V15_SHA224,

    AX_CIPHER_RSA_SIGN_RSASSA_PKCS1_V15_SHA256,

    AX_CIPHER_RSA_SIGN_RSASSA_PKCS1_V15_SHA384,

    AX_CIPHER_RSA_SIGN_RSASSA_PKCS1_V15_SHA512,

    AX_CIPHER_RSA_SIGN_RSASSA_PKCS1_PSS_SHA1,

    AX_CIPHER_RSA_SIGN_RSASSA_PKCS1_PSS_SHA224,

    AX_CIPHER_RSA_SIGN_RSASSA_PKCS1_PSS_SHA256,

    AX_CIPHER_RSA_SIGN_RSASSA_PKCS1_PSS_SHA384

    AX_CIPHER_RSA_SIGN_RSASSA_PKCS1_PSS_SHA512

    AX_CIPHER_RSA_SIGN_SCHEME_INVALID  = 0xffffffff,

}AX_CIPHER_RSA_SIGN_SCHEME_E;
```

【成员】

成员名称	描述
AX_CIPHER_RSA_SIGN_RSASSA_PKCS1_V15_SHA1	PKCS#1 RSASSA_PKCS1_V15_SHA1 签名算法

成员名称	描述
AX_CIPHER_RSA_SIGN_RSASSA_PKCS1_V15_SHA224	PKCS#1 RSASSA_PKCS1_V15_SH2241 签名算法
AX_CIPHER_RSA_SIGN_RSASSA_PKCS1_V15_SHA256	PKCS#1 RSASSA_PKCS1_V15_SHA256 签名算法
AX_CIPHER_RSA_SIGN_RSASSA_PKCS1_V15_SHA384	PKCS#1 RSASSA_PKCS1_V15_SHA384 法(V1.7.0_PX 不支持)
AX_CIPHER_RSA_SIGN_RSASSA_PKCS1_V15_SHA512	PKCS#1 RSASSA_PKCS1_V15_SHA512 法(V1.7.0_PX 不支持)
AX_CIPHER_RSA_SIGN_RSASSA_PKCS1_PSS_SHA1	PKCS#1 RSASSA_PKCS1_PSS_SHA1 签名算法
AX_CIPHER_RSA_SIGN_RSASSA_PKCS1_PSS_SHA224	PKCS#1 RSASSA_PKCS1_PSS_SHA224 签名算法
AX_CIPHER_RSA_SIGN_RSASSA_PKCS1_PSS_SHA256	PKCS#1 RSASSA_PKCS1_PSS_SHA256 签名算法
AX_CIPHER_RSA_SIGN_RSASSA_PKCS1_PSS_SHA384	PKCS#1 RSASSA_PKCS1_PSS_SHA384 法(V1.7.0_PX 不支持)
AX_CIPHER_RSA_SIGN_RSASSA_PKCS1_PSS_SHA512	PKCS#1

成员名称	描述
	RSASSA_PKCS1_PSS_SHA512 法(V1.7.0_PX 不支持)

【注意事项】

无

【相关数据类型及接口】

无

AX_CIPHER_CTRL_T

【说明】

定义加解密的控制信息。

【定义】

```
typedef struct
{
    AX_CIPHER_ALGO_E alg;
    AX_CIPHER_MODE_E workMode;
    AX_U8 * pKey;
    AX_U32 keySize;
    AX_U8 * pIV;
} AX_CIPHER_CTRL_T;
```

【成员】

成员名称	描述
Alg	加密的算法
workMode	加密算法的工作模式
pKey	秘钥
keySize	秘钥长度
pIV	初始向量

【注意事项】

无

【相关数据类型及接口】

无

AX_CIPHER_HASH_CTL_T

【说明】

定义加解密的控制信息。

【定义】

```
typedef struct {
    AX_U8 *hmacKey;
    AX_U32 hmackeyLen;
    AX_CIPHER_ALGO_E hashType;
}AX_CIPHER_HASH_CTL_T;
```

【成员】

成员名称	描述
hmacKey	Hmac 的 key 指针
hmackeyLen	Hmac key 的长度，单位为字节
hashType	Hash 的类型

【注意事项】

如果使用 HMAC，hmacKey 和 hmackeyLen 不可以为 0。

【相关数据类型及接口】

无

AX_CIPHER_RSA_PRIVATE_KEY

【说明】

定义 RSA 私钥的数据结构体。

【定义】

```
typedef struct {
    AX_U32 hashBits;
    AX_U32 modulusBits;
    AX_U8 *modulusData;
    AX_U32 privateExponentBytes;
    AX_U8 *exponentData;
    AX_CIPHER_RSA_SIGN_SCHEME_E enScheme;
}AX_CIPHER_RSA_PRIVATE_KEY;
```

【成员】

成员名称	描述
hashBits	进行 RSA 签名的 hash 的 bit
modulusBits	RSA 计算的私钥 modulus 的长度，单位 bit
modulusData	RSA 计算的私钥 modulus 数据指针
privateExponentBytes	私钥的 E 的长度
exponentData	私钥的 E 的指针
enScheme	RSA 计算的算法类型

【注意事项】

无

【相关数据类型及接口】

无

AX_CIPHER_RSA_PUBLIC_KEY

【说明】

定义 RSA 公钥的数据结构体。

【定义】

```
typedef struct {
    AX_U32 hashBits;
    AX_U32 modulusBits;
    AX_U8 *modulusData;
    AX_U32 publicExponentBytes;
    AX_U8 *exponentData;
    AX_CIPHER_RSA_SIGN_SCHEME_E enScheme;
}AX_CIPHER_RSA_PUBLIC_KEY;
```

【成员】

成员名称	描述
hashBits	进行 RSA 验证的 hash 的 bit
modulusBits	RSA 计算的公钥 modulus 的长度，单位 bit
modulusData	RSA 计算的公钥 modulus 数据指针
privateExponentBytes	公钥的 E 的长度
exponentData	公钥的 E 的指针
enScheme	RSA 计算的算法类型

【注意事项】

无

【相关数据类型及接口】

无

AX_CIPHER_RSA_ENC_SCHEME_E

【说明】

定义 RSA 加解密的数据填充方式。

【定义】

```
typedef enum

{

    AX_CIPHER_RSA_ENC_SCHEME_NO_PADDING,

    AX_CIPHER_RSA_ENC_SCHEME_PKCS1_V1_5,

} AX_CIPHER_RSA_ENC_SCHEME_E;
```

【成员】

成员名称	描述
AX_CIPHER_RSA_ENC_SCHEME_NO_PADDING	不填充
AX_CIPHER_RSA_ENC_SCHEME_PKCS1_V1_5	PKCS#1_V1_5 填充方式

【注意事项】

目前的版本不支持 OAEP 的填充方式，后续会增加扩展。

【相关数据类型及接口】

无

AX_CIPHER_RSA_PUB_ENC_T

【说明】

定义 RSA 加解密的公钥结构体。

【定义】

```
typedef struct {
    AX_CIPHER_RSA_ENC_SCHEME_E enScheme;
    AX_CIPHER_RSA_PUBLIC_KEY pubKey;
} AX_CIPHER_RSA_PUB_ENC_T;
```

【成员】

成员名称	描述
enScheme	RSA 加解密的填充方式
pubKey	RSA 加解密时的公钥信息

【注意事项】

无

【相关数据类型及接口】

无

AX_CIPHER_RSA_PRI_ENC_T

【说明】

定义 RSA 加解密时私钥的结构体。

【定义】

```
typedef struct {
    AX_CIPHER_RSA_ENC_SCHEME_E enScheme;
    AX_CIPHER_RSA_PRIVATE_KEY priKey;
} AX_CIPHER_RSA_PRI_ENC_T;
```

【成员】

成员名称	描述
enScheme	RSA 加解密的填充方式
priKey	RSA 加解密时的公钥结构体

【注意事项】

无

【相关数据类型及接口】

无

AX_CIPHER_SIG_DATA_T

【说明】

定义签名的数据的结构体。

【定义】

```
typedef struct

{

    AX_U8 * data;

    AX_U32 len;            // Data size in bytes

} AX_CIPHER_SIG_DATA_T;
```

【成员】

成员名称	描述
data	待签名的数据的指针
len	待签名的数据的长度

【注意事项】

无

【相关数据类型及接口】

无

4 错误码

错误码详见《55 - AX 软件错误码文档》文档。

5 调试信息

【调试信息】

```
# cat /proc/ax_proc/cipher
```

CIPHER 算子执行过程中，可以 cat 该节点，查看 CIPHER 状态，示例信息如下：

--------AES\DES\3DES\HASH\TRNG INFO--------

ALGO	MODE	ENCRYPT	KEY_LEN(B)	MIN_BW(MB\s)	AVG_BW(MB\s)	MAX_BW(MB\s)	COMPLETED	RUNNING
SHA1	-	0	0	0	0	7	37	0
AES	CBC	1	16	2	127	150	173	0
AES	ECB	1	16	1	136	155	172	0
AES	ECB	1	24	3	113	131	172	0
AES	ECB	1	32	3	97	113	172	0
AES	CBC	1	24	2	110	127	172	0
AES	CBC	1	32	3	94	110	172	0
AES	CTR	1	16	3	136	155	172	0
AES	CTR	1	24	2	113	131	172	0
AES	CTR	1	32	3	97	113	172	0
AES	ICM	1	16	2	128	155	344	0
AES	ECB	0	16	3	136	155	172	0
AES	ECB	0	24	3	113	131	172	0
AES	ECB	0	32	3	97	113	172	0
AES	CBC	0	16	3	136	155	172	0
AES	CBC	0	24	3	113	131	172	0
AES	CBC	0	32	3	97	113	172	0
AES	CTR	0	16	3	136	155	172	0
AES	CTR	0	24	3	113	131	172	0
AES	CTR	0	32	3	97	113	172	0
AES	ICM	0	16	2	128	155	344	0
DES	ECB	1	8	0	123	146	513	0
DES	CBC	1	8	1	133	146	171	0
3DES	ECB	1	24	1	37	50	513	0
3DES	CBC	1	24	1	39	50	342	0
DES	ECB	0	8	0	123	146	513	0
DES	CBC	0	8	1	133	146	171	0
3DES	ECB	0	24	1	37	50	513	0
3DES	CBC	0	24	1	39	50	342	0
SHA224	-	0	0	0	0	7	34	0
SHA256	-	0	0	0	0	7	34	0
SHA512	-	0	0	0	1	7	6	0
HMAC_SHA1	-	0	3	0	1	6	7	0
HMAC_SHA224	-	0	3	0	1	6	7	0
HMAC_SHA256	-	0	3	0	1	6	7	0
HMAC_SHA512	-	0	3	0	1	6	6	0
TRNG	-	0	0	0	10	32	10	0

----------RSA INFO----------

ALGO	KEY_LEN(B)	MIN_TIME(us)	AVG_TIME(us)	MAX_TIME(us)	COMPLETED	RUNNING
RSA_VERI_PKCS1_PSS_SHA1	128	403	403	403	2	0
RSA_VERI_PKCS1_PSS_SHA224	128	393	393	393	2	0
RSA_VERI_PKCS1_PSS_SHA256	128	366	366	366	2	0
RSA_VERI_PKCS1_PSS_SHA1	256	1197	1197	1198	2	0
RSA_VERI_PKCS1_PSS_SHA224	256	1227	1227	1227	2	0
RSA_VERI_PKCS1_PSS_SHA256	256	1163	1163	1163	2	0
RSA_VERI_PKCS1_PSS_SHA1	384	2542	2749	2956	2	0
RSA_VERI_PKCS1_PSS_SHA224	384	2537	2562	2587	2	0

【调试信息分析】

记录当前 CIPHER 算子调用信息。

【参数说明】

参数		描述
AES\DES\3DES\HASH\TRGN INFO **AES\DES\3DES\HASH\TRGN 等算法信息**	ALGO	算法类型
	MODE	工作模式
	ENCRYPT	是否是加密过程
	KEY_LEN	密码长度（单位：B）
	MIN_BW	最小带宽（单位：MB\s）
	AVG_BW	平均带宽（单位：MB\s）
	MAX_BW	最大带宽（单位：MB\s）
	COMPLETED	已经完成的任务个数
	RUNNING	正在执行的任务个数
RSA INFO **RSA 签名算法信息**	ALGO	算法类型
	KEY_LEN	密钥长度（单位：B）
	MIN_TIME	最小执行时间（单位：us）
	AVG_TIME	平均执行时间（单位：us）
	MAX_TIME	最大执行时间（单位：us）
	COMPLETED	已经完成的任务个数
	RUNNING	正在执行的任务个数

注意：如果数据量很小时，会导致算法带宽很低是正常现象。