XHSC 小华半导体
XIAOHUA SEMICONDUCTOR

32 位微控制器

CRYPTO_PKE 算法库使用

应用笔记

Rev1.01　2025 年 06 月

适用对象

产品系列	产品型号
HC32F4A8	ALL
HC32A4A8	ALL

声　明

★ 小华半导体有限公司（以下简称：“XHSC”）保留随时更改、更正、增强、修改小华半导体产品和/或本文档的权利，恕不另行通知。用户可在下单前获取最新相关信息。XHSC 产品依据购销基本合同中载明的销售条款和条件进行销售。

★ 客户应针对您的应用选择合适的 XHSC 产品，并设计、验证和测试您的应用，以确保您的应用满足相应标准以及任何安全、安保或其它要求。客户应对此独自承担全部责任。

★ XHSC 在此确认未以明示或暗示方式授予任何知识产权许可。

★ XHSC 产品的转售，若其条款与此处规定不同，XHSC 对此类产品的任何保修承诺无效。

★ 任何带有“®”或“™”标识的图形或字样是 XHSC 的商标。所有其他在 XHSC 产品上显示的产品或服务名称均为其各自所有者的财产。

★ 本通知中的信息取代并替换先前版本中的信息。

目　录

1 概述

本文档主要介绍小华半导体 HC32F4A8/ HC32A4A8 系列芯片 PKE 外设搭配 CryptoPKE 算法库实现 RSA、ECC、SM2 算法。

2 PKE 外设功能介绍

本系列产品的 PKE(Public Key Engine)系统由 1 个运算模块、2 个专用 RAM 存储器构成。配合软件驱动可以实现 RSA(RSA1024/ RSA2048/ RSA3072/ RSA4096)、ECC(brainpoolp 160/ brainpoolp 256/ brainpoolp 512/ SECP 192/ SECP 224/ SECP 256/ SECP 384/ SECP 521)、SM2 算法。

2.1 PKE 主要特性

2.1.1 支持运算

PKE 模块将 CPU 从复杂的公钥密码运算中解放了出来。CPU 只需要将输入参数配置好，PKE 就会根据目前的配置完成指定的操作。

- 支持模运算：模加、模减、模乘、模幂、模逆
- 支持椭圆曲线点运算：点加、点倍、点乘、验证点是否在曲线上
- 支持大数运算：大数乘法

3 算法库应用说明

3.1 CryptoPKE 算法库简介

CryptoPKE 算法库用于 PKE 外设实现 RSA、ECC、SM2 等算法。算法库采用 lib 库方式提供给用户免费使用，用户根据提供的函数接口实现对应的功能。

3.2 使用约定

3.2.1 检查 API 返回值

为了确保及时检测到异常（异常可能是用户输入不合法，也可能是攻击者攻击，或者其他原因造成），用户务必要检查调用的 API 返回值，对异常返回要有对应的处理。

3.2.2 大整数表示

公钥算法都会涉及到大整数运算。在算法库内部处理中，大整数被表示为 uint32_t 类型的小端数组。例如，若有大整数 a，其值为 0x998877665544332211，则需要三个 word 来存储，即表示如下：
uint32_t a[3] = {0x44332211, 0x88776655, 0x00000099};

RSA 算法提供的接口，涉及到的大数均是直接如此表示。而其他算法提供的接口，已经按算法标准要求表示了大数，用户无需进行直接的大数操作，具体的，对 SM2、ECDSA、ECDH 提供的接口里，大数均是字节大端表示。例如，同样是上面的例子，大整数 a 用字节大端表示是：

uint8_t a[9] = {0x99, 0x88, 0x77, 0x66, 0x55, 0x44, 0x33, 0x22, 0x11};

3.3 算法库依赖其它模块

3.3.1 硬件对接

这里支持的公钥算法，某些功能需要其他算法模块才能正常运行。例如，密钥对生成接口里，会调用到真随机数生成（TRNG）接口来实时生成随机数，作为密钥材料；SM2 算法的加解密，密钥协商会调用 SM3 算法计算 hash 值。

注：

1) 如果使用硬件随机数需要用户实现 *TRNG* 外设初始化并将获取随机数接口注册到算法库，否则默认使用算法库内部软件方式生成随机数。

2) 为了算法兼容性算法库内部均已软件实现必要的 *hash* 算法，用户无需进行硬件对接。

3.4 RSA

3.4.1 概述

RSA 算法是第一个既可用于数字签名又能用于加解密的公钥算法。RSA 算法可分为原始形式和 CRT 形式（为以示区别，原始形式也称作非 CRT 形式），前者即最初提出 RSA 算法的形式，CRT 形式是为了加速 RSA 私钥运算发展而来的 RSA 算法变体，其优势是性能得到提高，一般会是原始形式私钥运算的 3 倍到 4 倍之间。

算法库支持两种形式的 RSA 密钥对生成，以及两种形式的模幂（普通模幂与 CRT 模幂）。

3.4.2 接口说明

表 3-1 RSA 函数接口

函数名	描述
uint32_t RSA_ModExp(uint32_t *a, uint32_t *e, uint32_t *n, uint32_t *out, uint32_t eBitLen, uint32_t nBitLen)	RSA 模幂运算，可用于非 CRT 形式 RSA 的签名/验签/加密/解密/，以及 CRT 形式的验签/加密
uint32_t RSA_CRTModExp(uint32_t *a, uint32_t *p, uint32_t *q, uint32_t *dp, uint32_t*dq, uint32_t *u, uint32_t *out, uint32_t nBitLen)	RSA CRT 模幂运算，可用于 CRT 形式 RSA 的签名/解密
uint32_t RSA_GetKey(uint32_t *e, uint32_t *d, uint32_t *n, uint32_t eBitLen, uint32_t nBitLen)	非 CRT 形式 RSA 密钥对生成
uint32_t RSA_GetCRTKey(uint32_t *e, uint32_t *p, uint32_t *q, uint32_t *dp, uint32_t *dq, uint32_t *u, uint32_t *n, uint32_t eBitLen, uint32_t nBitLen)	CRT 形式 RSA 密钥对生成

1） **RSA_ModExp**

函数名	RSA_ModExp
函数原型	uint32_t RSA_ModExp(uint32_t *a, uint32_t *e, uint32_t *n, uint32_t *out, uint32_t eBitLen, uint32_t nBitLen)
功能	RSA 模幂运算
输入	uint32_t *a 底数 uint32_t *e 指数 uint32_t *n 公共模数 uint32_t eBitLen 指数 e 的比特长度 uint32_t nBitLen 模数 n 的比特长度
输出	uint32_t *out out = $a^e \bmod n$
返回值	RSA_SUCCESS 成功 Other 失败，详见错误码定义
注意事项	1. a，e，n，out 的实际比特长度都不能大于 RSA_MAX_BIT_LEN 2. nBitLen 不能是奇数，且不能大于 RSA_MAX_BIT_LEN 3. eBitLen 不要超过 nBitLen 4. a 要小于 n，且 a 和 n 都占用(nBitLen+31)/32 个字的空间，没有用到的高位要填充 0

	5. e 占用(eBitLen+31)/32 个字的空间，没有用到的高位要填充 0

2) RSA_CRTModExp

函数名	RSA_CRTModExp
函数原型	uint32_t RSA_CRTModExp(uint32_t *a, uint32_t *p, uint32_t *q, uint32_t *dp, uint32_t *dq, uint32_t *u, uint32_t *out, uint32_t nBitLen)
功能	RSA CRT 模幂运算
输入	uint32_t *a 底数 uint32_t *p 私钥 p uint32_t *q 私钥 q uint32_t *dp 私钥 dp uint32_t *dq 私钥 dq uint32_t *u 私钥 u $=q^{-1} \bmod p$ uint32_t nBitLen 模数 n 的比特长度
输出	uint32_t *out 输出，即私钥运算结果
返回值	RSA_SUCCESS 成功 Other 失败，详见错误码定义
注意事项	1. a，n，out 的实际比特长度都不能大于 RSA_MAX_BIT_LEN 2. nBitLen 不能是奇数，且不能大于 RSA_MAX_BIT_LEN 3. a 要小于 n，且 a 和 n 都占用(nBitLen+31)/32 个字的空间，没有用到的高位要填充 0 4. p,q,dp,dq,u 占用的字的个数都是(nBitLen/2+31)/32，没有用到的高位要填充 0

3) RSA_GetKey

函数名	RSA_GetKey
函数原型	uint32_t RSA_GetKey(uint32_t *e, uint32_t *d, uint32_t *n, uint32_t eBitLen, uint32_t nBitLen)
功能	非 CRT 形式 RSA 密钥对生成
输入	uint32_t eBitLen 公钥 e 的比特长度 uint32_t nBitLen 公共模数 n 的比特长度
输出	uint32_t *e 公钥 e uint32_t *d 私钥 d uint32_t *n 公共模数 n
返回值	RSA_SUCCESS 成功 Other 失败，详见错误码定义
注意事项	1. nBitLen 不能是奇数，且不能大于 RSA_MAX_BIT_LEN 2. 1<eBitLen<=nBitLen

	3. 当 eBitLen 取值为 2/5/17 时，对应的 e 的值固定是 3/17/65537，其余情况则 e 取指定比特长度的随机值 4. 请确保 e,d,n 各自的空间足够，其中 e 占用的字的个数是(eBitLen+31)/32，d 和 n 占用的字的个数都是(nBitLen+31)/32

4) **RSA_GetCRTKey**

函数名	RSA_GetCRTKey
函数原型	uint32_t RSA_GetCRTKey(uint32_t *e, uint32_t *p, uint32_t *q, uint32_t *dp, uint32_t *dq, uint32_t *u, uint32_t *n, uint32_t eBitLen, uint32_t nBitLen)
功能	CRT 形式 RSA 密钥对生成
输入	uint32_t eBitLen 公钥 e 的比特长度 uint32_t nBitLen 公共模数 n 的比特长度
输出	uint32_t *e 公钥 e uint32_t *p 私钥 p uint32_t *q 私钥 q uint32_t *dp 私钥 dp uint32_t *dq 私钥 dq uint32_t *u 私钥 u uint32_t *n 公共模数 n
返回值	RSA_SUCCESS 成功 Other 失败，详见错误码定义
注意事项	1. nBitLen 不能是奇数，且不能大于 RSA_MAX_BIT_LEN 2. 1<eBitLen<=nBitLen 3. 当 eBitLen 取值为 2/5/17 时，对应的 e 的值固定是 3/17/65537，其余情况则 e 取指定比特长度的随机值 4. 请确保 e,p,q,dp,dq,u,n 各自的空间足够，其中 e 占用的字的个数是(eBitLen+31)/32，n 占用的字的个数都是(nBitLen+31)/32，p,q,dp,dq,u 占用的字的个数都是(nBitLen/2+31)/32

3.4.3 例程介绍

本文主要以 pke_rsa（applications/ crypto/ pke_rsa）样例进行介绍，该样例展示了 PKE 外设使用 CryptoPKE 算法库进行 RSA 算法的加密、解密以及签名、验签。

1) **基本流程**

RSA 算法加密、解密流程如下图 3-1 所示：

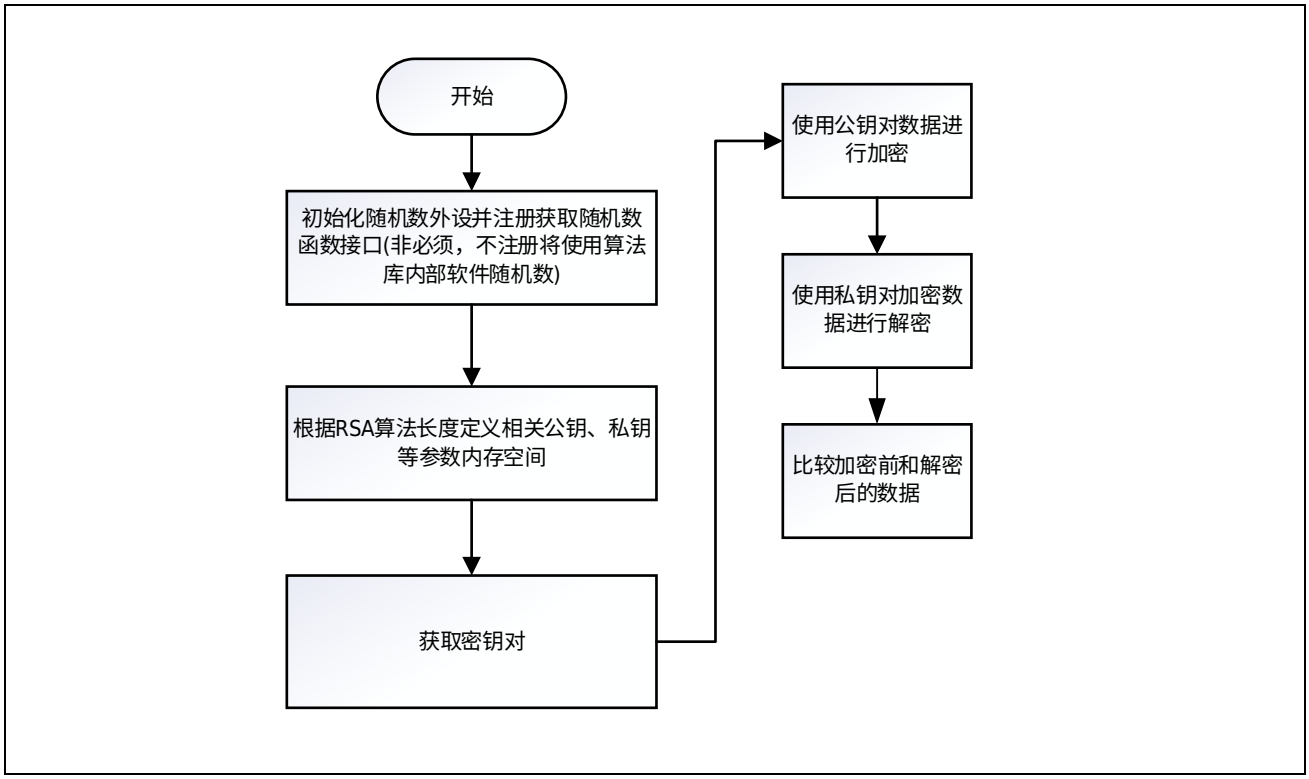

图 3-1 RSA 加密、解密基本流程

RSA 算法签名、验签流程如下图 3-2 所示：

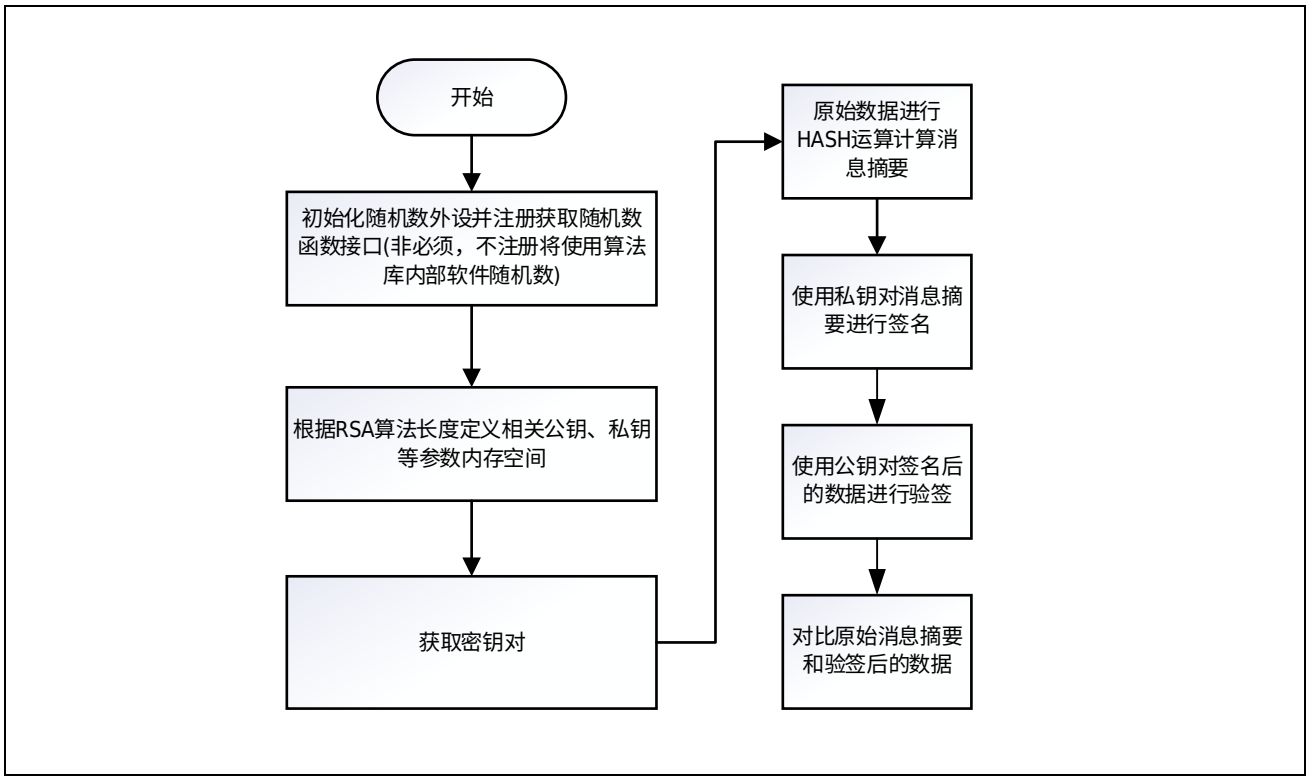

图 3-2 RSA 签名、验签基本流程

2) **源码说明**

随机数初始化与注册：

```
/* Trng configuration and register(not neccessary if using software trng) */
TrngConfig();
hw_trng_register(HC32_Get_Rand);
```

封装 HC32 获取随机数的函数：

```
/**
 * @brief    Get random number
 * @param    [out] rand          Points to the random number buff
 * @param    [in] bytes          Random number length
 * @retval uint32_t:
 *                - LL_OK:                      No error occurred.
 *                - LL_ERR_TIMEOUT:             Works timeout.
 *                - LL_ERR_INVD_PARAM:          pu32Random == NULL or u8RandomLen == 0
 */
static uint32_t HC32_Get_Rand(uint8_t *rand, uint32_t bytes)
{
    uint32_t ret;
    uint32_t Temp;
    if (bytes < 4) {
```

```
            ret = TRNG_GenerateRandom(&Temp, 1);
            for (;bytes > 0; bytes--) {
                rand[bytes - 1] = (uint8_t)Temp;
                Temp >>= 8;
            }
        } else {
            ret = TRNG_GenerateRandom((uint32_t *)rand, bytes / 4);
        }

        return ret;
    }
```

算法库内部使用的获取随机数函数接口与 HC32 获取随机数接口不兼容，因此需要做接口转换处理。HC32 接口获取数据为 uint32 类型，算法库为 uint8 类型，因此需要将算法库参数的长度/4 作为 HC32 的数据长度，同时需要对数据长度小于 4 情况做单独处理。

注：

如果使用硬件随机数需要用户实现 *TRNG* 外设初始化并将获取随机数接口注册到算法库，否则默认使用算法库内部软件方式生成随机数。

算法参数定义：

```
#define RSA_BIT_LEN                     (1024U)
```

RSA_BIT_LEN 用于定义 RSA 算法长度，例如 1024/ 2048/ 3072/ 4096，长度不能是奇数，且不能大于 RSA_MAX_BIT_LEN(4096)

```
uint32_t e[RSA_BIT_LEN/32] = {0};
uint32_t d[RSA_BIT_LEN/32] = {0};
uint32_t n[RSA_BIT_LEN/32] = {0};

uint32_t p[RSA_BIT_LEN/32/2] = {0};
uint32_t q[RSA_BIT_LEN/32/2] = {0};
uint32_t dp[RSA_BIT_LEN/32/2] = {0};
uint32_t dq[RSA_BIT_LEN/32/2] = {0};
uint32_t u[RSA_BIT_LEN/32/2] = {0};

uint32_t in[RSA_BIT_LEN/32] = {0};
uint32_t ciphertext[RSA_BIT_LEN/32] = {0};
uint32_t plaintext[RSA_BIT_LEN/32] = {0};

uint32_t eBitLen = 17;                          /* e = 65537 */
uint32_t nBitLen = RSA_BIT_LEN;
uint32_t nWordLen = GET_WORD_LEN(nBitLen);
```

1) 由于 RSA_BIT_LEN 长度为 Bit 长度，实际参数为 uint32 类型，因此实际参数长度等于 RSA_BIT_LEN/32

2) 由于 n = pq，因此 p,q 等参数数据长度均为 n 的一半，即 RSA_BIT_LEN/32/2

3) eBitLen = 17 表示 e 取值为 65537，因为 65537 二进制表示需要 17bit

NON-CRT 形式加密、解密：

```
/* get key pair */
ret = RSA_GetKey(e, d, n, eBitLen, nBitLen);
if (RSA_SUCCESS != ret) {
    return PKE_ERROR;
}

/* raw data */
msg = "NON-CRT encrypt & decrypt test";
memset(in, 0, nWordLen);
memcpy(in, msg, strlen(msg));

/* public key encrypt */
ret = RSA_ModExp(in, e, n, ciphertext, eBitLen, nBitLen);
if (RSA_SUCCESS != ret) {
    return PKE_ERROR;
}

/* private key decrypt */
ret = RSA_ModExp(ciphertext, d, n, plaintext, get_valid_bits(d, nWordLen), nBitLen);
if (RSA_SUCCESS != ret) {
    return PKE_ERROR;
}

/* compare data */
if (memcmp(in, plaintext, nWordLen)) {
    return PKE_ERROR;
}
```

1) 生成公钥和私钥

2) 初始化原始数据

3) 使用公钥 e 和 n 对原始数据进行加密，加密运算是$C = M^e \ mod \ n$

4) 使用私钥 d 和 n 对加密数据进行解密，解密运算是$M' = C^d \ mod \ n$

5) 对比原始数据和解密后数据

NON-CRT 形式签名、验签：

```
/* get key pair */
ret = RSA_GetKey(e, d, n, eBitLen, nBitLen);
```

```
    if (RSA_SUCCESS != ret) {
        return PKE_ERROR;
    }

    /* raw data */
    msg = "NON-CRT sign & verify test";
    /* hash calculate */
    ret = hash_init(ctx, HASH_SHA256);
    if(HASH_SUCCESS != ret) {
        return HASH_ERROR;
    }
    ret = hash_update(ctx, (const uint8_t *)msg, strlen(msg));
    if(HASH_SUCCESS != ret) {
        return HASH_ERROR;
    }
    memset(in, 0, nWordLen);
    ret = hash_final(ctx, (uint8_t *)in);
    if(HASH_SUCCESS != ret) {
        return HASH_ERROR;
    }
    /* big endian -> little endian */
    /* SHA256 = 256/32 Word */
    for(i=0; i<(256/32); i++) {
        in[i] = __REV(in[i]);
    }

    /* private key sign */
    ret = RSA_ModExp(in, d, n, sign_out, get_valid_bits(d, nWordLen), nBitLen);
    if (RSA_SUCCESS != ret) {
        return PKE_ERROR;
    }

    /* public key verify */
    ret = RSA_ModExp(sign_out, e, n, verify_out, eBitLen, nBitLen);
    if (RSA_SUCCESS != ret) {
        return PKE_ERROR;
    }

    /* compare data */
    if (memcmp(in, verify_out, nWordLen)) {
        return PKE_ERROR;
    }
```

1) 生成公钥和私钥

2) 初始化原始数据并将原始数据做 HASH 运算得到消息摘要（根据需要做大小端转换处理）

3) 使用私钥 d 和 n 进行签名，签名运算是$S = M^d \ mod \ n$

4) 使用公钥 e 和 n 进行验签，验证签名运算是$M' = S^e \ mod \ n$

5) 对比原始消息摘要数据和验签后数据

注：

CRT 形式的 *RSA* 算法步骤同 *NON-CRT* 形式算法步骤一致。

3.4.4 Openssl 交叉验证

使用 Openssl 生成的密钥需要转换成 CryptoPKE 算法库使用的公钥 e、公共模数 n 等形式，转换方式根据使用的 Openssl 编程语言有所不同需要用户自行实现。

公钥算法都会涉及到大整数运算。在算法库内部处理中，大整数被表示为 uint32_t 类型的小端数组。

因此，假如大整数 n 在 PC 端其值为 0xccbbaa998877665544332211，则在 MCU 端则需要转换为 0x4433221188776655ccbbaa99，转换示例代码如下：

```
char *text_n = “ccbbaa998877665544332211”;
/* big endian array -> little endian array */
char dest[8];
for(j=0; j<nWordLen; j++) {
    strncpy(dest, text_n+(nWordLen-1-j)*8, 8);
    sscanf(dest, "%08x", &n[j]);
}
```

其中 text_n 为指向待转换 n 的十六进制字符串。

注：

word 的高位不存在则必须填充 *0*，否则会被当作有效的值。

3.5 ECC

ECC 英文全称“Elliptic Curve Cryptography”，其背后的密码学原理或者说安全性，是基于椭圆曲线离散对数问题（Elliptic Curve Discrete Logarithm Problem，ECDLP）。

3.5.1 ECDSA

ECDSA（Elliptic Curve Digital Signature Algorithm）算法是 DSA 签名算法的椭圆曲线版本。ECDSA 使用的椭圆曲线方程为$y^2 = x^3 + ax + b$，支持这类曲线如 secp192r1、secp224r1、secp256r1、secp384r1、secp521r1、brainpoolP160r1、brainpoolP192r1、brainpoolP256r1、brainpoolP320r1、brainpoolP384r1、brainpoolP512r1 等。

CryptoPKE 算法库支持 ECDSA 的密钥对生成，签名和验签。

3.5.1.1 接口说明

表 3-2 ECDSA 函数接口

函数名	描述
uint32_t eccp_get_pubkey_from_prikey (eccp_curve_t *curve, uint8_t *priKey, uint8_t *pubKey)	从私钥获得公钥
uint32_t eccp_getkey (eccp_curve_t *curve, uint8_t *priKey, uint8_t *pubKey)	生成随机密钥对
uint32_t ecdsa_sign (eccp_curve_t *curve, uint8_t *E, uint32_t EByteLen, uint8_t *rand_k, uint8_t *priKey, uint8_t *signature)	ECDSA 签名生成
uint32_t ecdsa_verify (eccp_curve_t *curve, uint8_t *E, uint32_t EByteLen, uint8_t *pubKey, uint8_t *signature)	ECDSA 签名验证

1) **eccp_get_pubkey_from_prikey**

函数名	eccp_get_pubkey_from_prikey
函数原型	uint32_t eccp_get_pubkey_from_prikey (eccp_curve_t *curve, uint8_t *priKey, uint8_t *pubKey)
功能	由私钥生成椭圆曲线公钥
输入	eccp_curve_t *curve 椭圆曲线结构体 eccp_curve_t 指针 uint8_t *priKey 私钥，字节大端表示
输出	uint8_t *pubKey 公钥，按顺序分别是 x 坐标和 y 坐标，字节大端表示
返回值	PKE_SUCCESS 成功 Other 失败，详见错误码定义
注意事项	1. 该接口声明在 pke.h 中，可用于 ECDSA 和 ECDH 的从私钥生成公钥。 2. 大数占用的字节数为曲线参数 p 或 n 占用的字节数，例如，若曲线是 secp521r1，那么私钥占用是 (521+7)/8=66 字节，公钥占用是 2*66=132 字节。

2) **eccp_getkey**

函数名	eccp_getkey
函数原型	uint32_t eccp_getkey(eccp_curve_t *curve, uint8_t *priKey, uint8_t *pubKey)
功能	椭圆曲线随机密钥对生成
输入	eccp_curve_t *curve 椭圆曲线结构体 eccp_curve_t 指针
输出	uint8_t *priKey 私钥，字节大端表示 uint8_t *pubKey 公钥，按顺序分别是 x 坐标和 y 坐标，字节大端表示
返回值	PKE_SUCCESS 成功 Other 失败，详见错误码定义
注意事项	1. 该接口声明在 pke.h 中，可用于 ECDSA 和 ECDH 的从私钥生成公钥。 2. 大数占用的字节数为曲线参数 p 或 n 占用的字节数，例如，若曲线是 secp521r1，那么私钥占用是 (521+7)/8=66 字节，公钥占用是 2*66=132 字节。

3) **ecdsa_sign**

函数名	ecdsa_sign
函数原型	uint32_t ecdsa_sign (eccp_curve_t *curve,

	uint8_t *E, uint32_t EByteLen, uint8_t *rand_k, uint8_t *priKey, uint8_t *signature)
功能	ECDSA 签名
输入	eccp_curve_t *curve 椭圆曲线结构体 eccp_curve_t 指针 uint8_t *E 待签名的 hash 值 uint32_t EByteLen hash 值的字节长度 uint8_t *rand_k 签名中的随机数 k，字节大端表示；若无则赋值 NULL 即可，函数内部会自行实时生成 uint8_t *priKey 私钥，字节大端表示
输出	uint8_t *signature 签名，按顺序由两个大数 r 和 s 组成，字节大端表示
返回值	ECDSA_SUCCESS 成功 Other 失败，详见错误码定义
注意事项	1. 通常对消息的 hash 值进行签名和验签,而从 hash 值 E 得到签名或验签中的输入大数 e,是基于 SEC1 V2 或 ANSI X9.62 中的方法。具体的，若 hash 值 E 字节长度小于所选曲线的生成元的阶 n 的字节长度，则 E 值直接转换为大数 e；而若 E 值长度大于等于 n 的长度，则从 E 值最高 bit 开始，截断 nbit 个 bit，这里 nbit 表示 n 的比特长度。 例如，若选取了曲线 secp521r1，E 值若是 66 字节或更长，则 66 字节之后的部分会被直接丢弃，而前 66 字节是 528bit，实际会截断高 521bit 作为转换后的大数 e 参与运算，低 7bit 也会被丢弃。 2. 若没有 rand_k，请将该参数设置为 NULL，函数内部会实时生成一个随机 rand_k 参与签名计算，这也是推荐的做法。 3. 大数占用的字节数为曲线参数 p 或 n 占用的字节数，例如，若曲线是 secp521r1，那么私钥占用是 (521+7)/8=66 字节，公钥占用是 2*66=132 字节，而签名是两个大数 r 和 s，故和公钥占用一样长。

4) ecdsa_verify

函数名	ecdsa_verify
函数原型	uint32_t ecdsa_verify (eccp_curve_t *curve, uint8_t *E, uint32_t EByteLen, uint8_t *pubKey, uint8_t *signature)
功能	ECDSA 验签
输入	eccp_curve_t *curve 椭圆曲线结构体 eccp_curve_t 指针 uint8_t *E 待签名的 hash 值 uint32_t EByteLen hash 值的字节长度 uint8_t *pubKey 公钥，按顺序分别是 x 坐标和 y 坐标，字节大端表示 uint8_t *signature 签名，按顺序由两个大数 r 和 s 组成，字节大端表示
输出	无
返回值	ECDSA_SUCCESS 成功 Other 失败，详见错误码定义
注意事项	1. 通常对消息的 hash 值进行签名和验签,而从 hash 值 E 得到签名或验签中的输入大数 e,是基于 SEC1 V2 或 ANSI X9.62 中的方法。具体的，若 hash 值 E 字节长度小于所选曲线的生成元的阶 n 的字节长度，

	则 E 值直接转换为大数 e；而若 E 值长度大于等于 n 的长度，则从 E 值最高 bit 开始，截断 nbit 个 bit，这里 nbit 表示 n 的比特长度。 例如，若选取了曲线 secp521r1，E 值若是 66 字节或更长，则 66 字节之后的部分会被直接丢弃，而前 66 字节是 528bit，实际会截断高 521bit 作为转换后的大数 e 参与运算，低 7bit 也会被丢弃。 2. 大数占用的字节数为曲线参数 p 或 n 占用的字节数，例如，若曲线是 secp521r1，那么私钥占用是 (521+7)/8=66 字节，公钥占用是 2*66=132 字节，而签名是两个大数 r 和 s，故和公钥占用一样长。

3.5.1.2 例程介绍

本文主要以 pke_ecdsa（applications/ crypto/ pke_ecdsa）样例进行介绍，该样例展示了 PKE 外设使用 CryptoPKE 算法库进行 ECDSA 算法的签名、验签。

1) 基本流程

ECDSA 算法签名、验签流程如下图 3-3 所示：

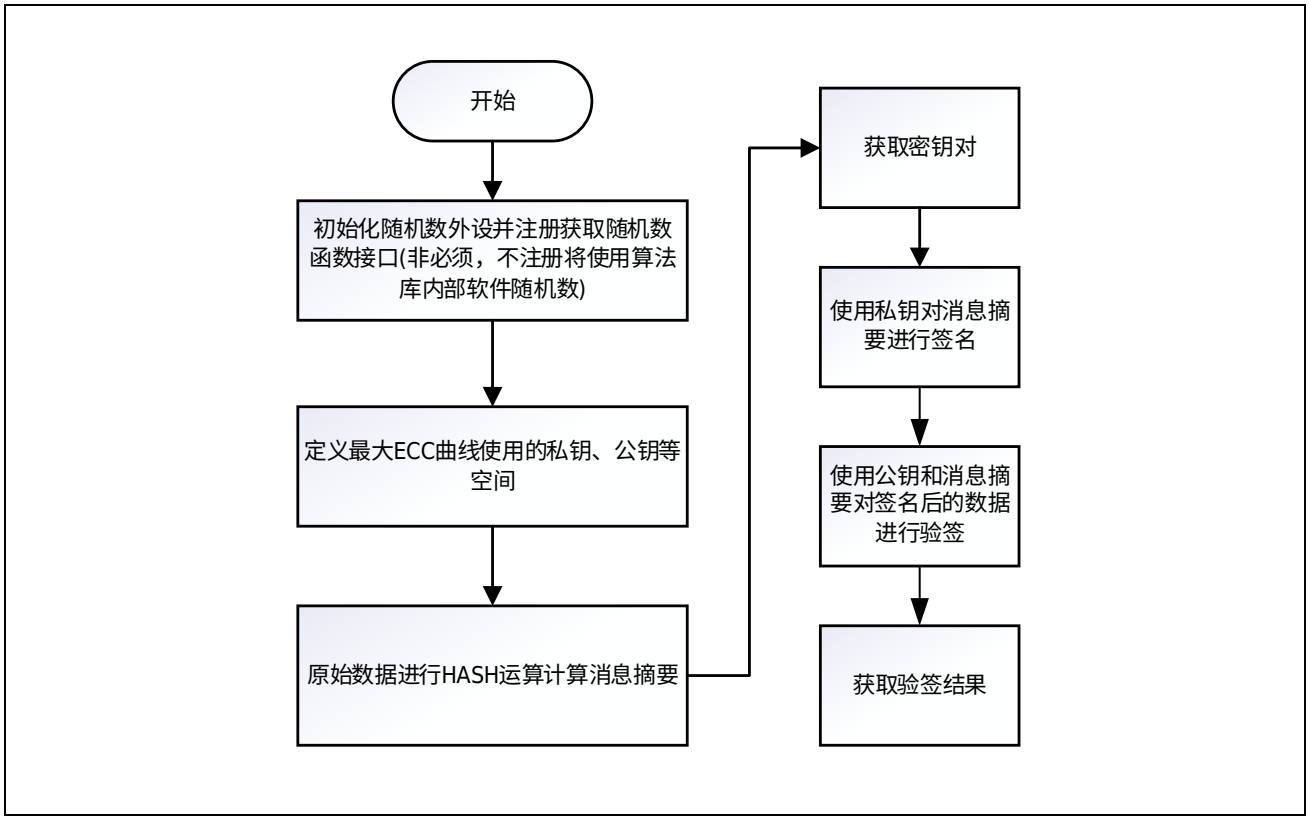

图 3-3 ECDSA 签名、验签基本流程

2) 源码说明

随机数初始化与注册：

参考 RSA 随机数初始化与注册说明部分。

算法参数定义：

```
#define E_MAX_BYTE_LEN                (64U)
```

E_MAX_BYTE_LEN 定义用于签名的 hash 值最大长度，若 hash 值 E 字节长度小于所选曲线的生成元的阶 n 的字节长度，则 E 值直接转换为大数 e；而若 E 值长度大于等于 n 的长度，则从 E 值最高 bit 开始，截断 nbit 个 bit，这里 nbit 表示 n 的比特长度。

```
uint8_t priKey[ECCP_MAX_BYTE_LEN] = {0};
uint8_t pubKey[2*ECCP_MAX_BYTE_LEN] = {0};        /* x+y */
uint8_t signature[2*ECCP_MAX_BYTE_LEN] = {0};     /* r+s */
uint8_t E[E_MAX_BYTE_LEN];
```

1) ECCP_MAX_BYTE_LEN 宏表示 ECC 支持的最大曲线参数

2) 公钥由 x、y 坐标组成，因此长度为私钥的 2 倍

3) 签名数据由两个大数 r 和 s 组成，因此长度为私钥的 2 倍

ECDSA 签名、验签：

```
/* raw data */
msg = "ECDSA sign & verify test";
/* hash calculate */
ret = hash_init(ctx, HASH_SHA256);
if(HASH_SUCCESS != ret) {
    return HASH_ERROR;
}
/* SHA256 = 256/8 Bytes */
EByteLen = 256U/8U;
ret = hash_update(ctx, (const uint8_t *)msg, strlen(msg));
if(HASH_SUCCESS != ret) {
    return HASH_ERROR;
}
memset(E, 0, EByteLen);
ret = hash_final(ctx, E);
if(HASH_SUCCESS != ret) {
    return HASH_ERROR;
}
/* little endian array -> big endian array */
for(i=0; i<EByteLen/4U; i++) {
    E[i] = __REV(E[i]);
}

/* Get key pair */
ret = eccp_getkey((eccp_curve_t *)curve, priKey, pubKey);
if(PKE_SUCCESS != ret) {
    return PKE_ERROR;
}
```

```
/* Sign the hash value using the private key */
ret = ecdsa_sign((eccp_curve_t *)curve, E, EByteLen, NULL, priKey, signature);
if(ECDSA_SUCCESS != ret) {
    return PKE_ERROR;
}

/* Verify the signature data using the public key */
ret = ecdsa_verify((eccp_curve_t *)curve, E, EByteLen, pubKey, signature);
if(ECDSA_SUCCESS != ret) {
    return PKE_ERROR;
}
```

1) 初始化原始数据并将原始数据做 HASH 运算得到消息摘要 E（根据需要做大小端转换处理）

2) 生成公钥和私钥

3) 使用私钥和 E 进行签名

4) 使用公钥和 E 对签名数据进行验签

3.5.1.3 Openssl 交叉验证

使用 Openssl 生成的公钥和签名数据等需要转换成 CryptoPKE 算法库使用的公钥坐标(x、y)，签名数据(大数 r、s)等形式，转换方式根据使用的 Openssl 编程语言有所不同需要用户自行实现。

注：

1. 由于 *CryptoPKE* 算法库内部默认已经采用字节大端方式，因此和 *PC* 端参数交换不必再做大小端转换处理。
2. *Openssl* 生成的公钥必须转换成 *x*、*y* 坐标的十六进制数据形式，且按照前一半数据为 *x* 坐标，后一半数据为 *y* 坐标顺序存放。
3. *Openssl* 生成的签名数据给 *CryptoPKE* 算法库进行验签必须转换成大数 *r*、*s* 的十六进制数据形式，且按照前一半数据为大数 *r*，后一半数据为大数 *s* 顺序存放。

3.5.2 ECDH

ECDH（Elliptic Curve Diffie–Hellman）算法是 DH 密钥协商（或密钥交换）算法的椭圆曲线版本。和 ECDSA 一样，ECDH 使用使用的椭圆曲线方程为$y^2 = x^3 + ax + b$，支持的这类曲线如 secp192r1、secp224r1、secp256r1、secp384r1、secp521r1、brainpoolP160r1、brainpoolP192r1、brainpoolP256r1、brainpoolP320r1、brainpoolP384r1、brainpoolP512r1 等。

CryptoPKE 算法库支持 ECDH 的密钥对生成，密钥协商计算。

3.5.2.1 接口说明

表 3-3 ECDH 函数接口

函数名	描述
uint32_t eccp_get_pubkey_from_prikey(eccp_curve_t *curve, uint8_t *priKey, uint8_t *pubKey)	从私钥获得公钥
uint32_t eccp_getkey(eccp_curve_t *curve, uint8_t *priKey, uint8_t *pubKey)	生成随机密钥对
uint32_t ecdh_compute_key(eccp_curve_t *curve, uint8_t *local_prikey, uint8_t *peer_pubkey, uint8_t *key,uint32_t keyByteLen, KDF_FUNC kdf)	ECDH 密钥协商计算

1) eccp_get_pubkey_from_prikey

函数名	eccp_get_pubkey_from_prikey
函数原型	uint32_t eccp_get_pubkey_from_prikey (eccp_curve_t *curve, uint8_t *priKey, uint8_t *pubKey)
功能	由私钥生成椭圆曲线公钥
输入	eccp_curve_t *curve 椭圆曲线结构体 eccp_curve_t 指针 uint8_t *priKey 私钥，字节大端表示
输出	uint8_t *pubKey 公钥，按顺序分别是 x 坐标和 y 坐标，字节大端表示
返回值	PKE_SUCCESS 成功 Other 失败，详见错误码定义
注意事项	1. 该接口声明在 pke.h 中，可用于 ECDSA 和 ECDH 的从私钥生成公钥。 2. 大数占用的字节数为曲线参数 p 或 n 占用的字节数，例如，若曲线是 secp521r1，那么私钥占用是 (521+7)/8=66 字节，公钥占用是 2*66=132 字节。

2) eccp_getkey

函数名	eccp_getkey
函数原型	uint32_t eccp_getkey (eccp_curve_t *curve, uint8_t *priKey, uint8_t *pubKey)
功能	椭圆曲线随机密钥对生成
输入	eccp_curve_t *curve 椭圆曲线结构体 eccp_curve_t 指针
输出	uint8_t *priKey 私钥，字节大端表示 uint8_t *pubKey 公钥，按顺序分别是 x 坐标和 y 坐标，字节大端表示
返回值	PKE_SUCCESS 成功 Other 失败，详见错误码定义
注意事项	1. 该接口声明在 pke.h 中，可用于 ECDSA 和 ECDH 的从私钥生成公钥。 2. 大数占用的字节数为曲线参数 p 或 n 占用的字节数，例如，若曲线是 secp521r1，那么私钥占用是 (521+7)/8=66 字节，公钥占用是 2*66=132 字节。

3) ecdh_compute_key

函数名	ecdh_compute_key
函数原型	uint32_t ecdh_compute_key(eccp_curve_t *curve, uint8_t *local_prikey, uint8_t *peer_pubkey,

	uint8_t *key, uint32_t keyByteLen, KDF_FUNC kdf)
功能	ECDH 密钥协商计算
输入	eccp_curve_t *curve 椭圆曲线结构体 eccp_curve_t 指针 uint8_t *local_prikey 己方私钥 uint8_t *peer_pubkey 对方公钥 uint32_t keyByteLen 协商密钥的字节长度 KDF_FUNC kdf 密钥导出函数，若无则赋值 NULL
输出	uint8_t *key 协商密钥
返回值	ECDH_SUCCESS 成功 Other 失败，详见错误码定义
注意事项	1. 大数占用的字节数为曲线参数 p 或 n 占用的字节数，例如，若曲线是 secp521r1，那么私钥占用是(521+7)/8=66 字节，公钥占用是 2*66=132 字节。 2. 若无密钥导出函数，则密钥协商的 key 直接从内部点乘结果的 x 坐标最高字节开始截取（x 坐标用字节大端表示），此时若 keyByteLen 大于 x 坐标占用的字节数，则实际只输出完整的 x 坐标。

3.5.2.2 例程介绍

本文主要以 pke_ecdh（applications/ crypto/ pke_ecdh）样例进行介绍，该样例展示了 PKE 外设使用 CryptoPKE 算法库进行 ECDH 算法的密钥协商(密钥交换)。

1) 基本流程

ECDH 算法密钥协商流程如下图 3-4 所示：

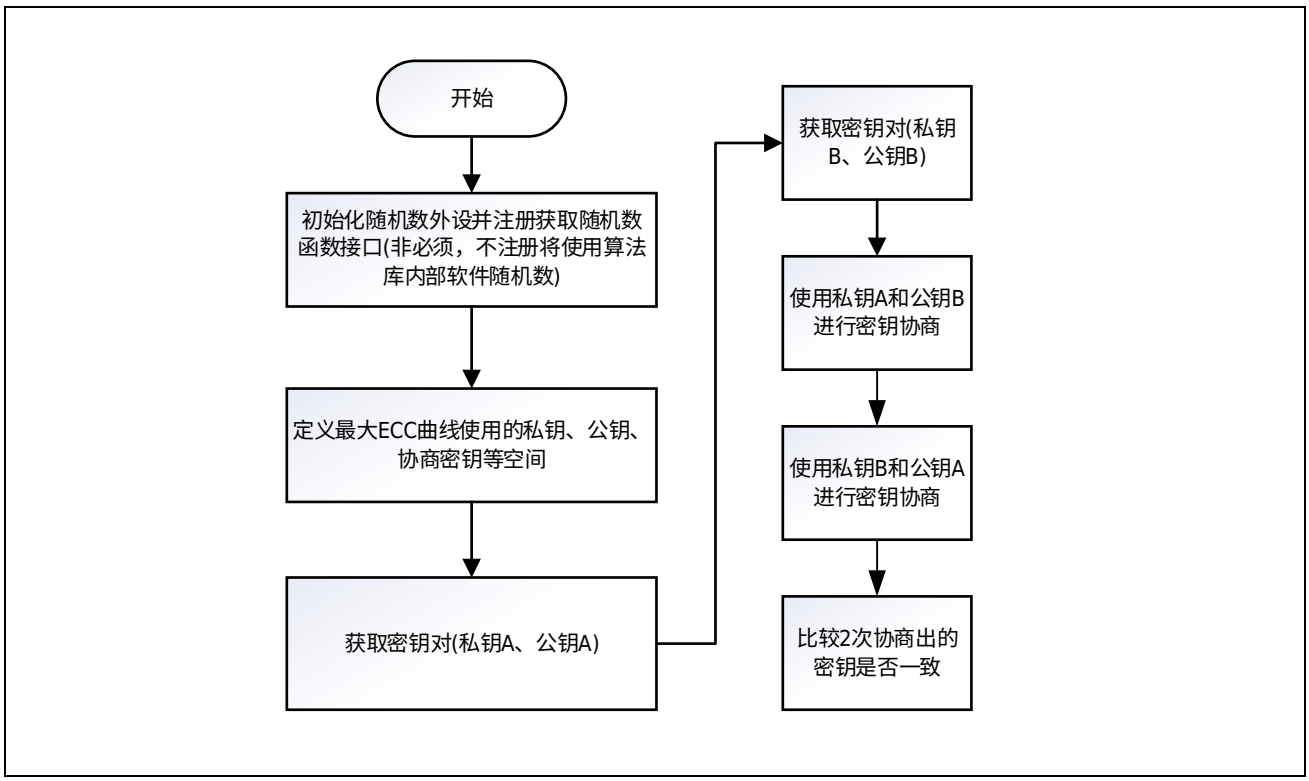

图 3-4 ECDH 密钥协商基本流程

2) 源码说明

随机数初始化与注册：

参考 RSA 随机数初始化与注册说明部分。

算法参数定义:

```
uint8_t Prikey_A[ECCP_MAX_BYTE_LEN] = {0};
uint8_t Pubkey_A[2*ECCP_MAX_BYTE_LEN] = {0};
uint8_t ComputeKey_A[ECCP_MAX_BYTE_LEN] = {0};
uint8_t Prikey_B[ECCP_MAX_BYTE_LEN] = {0};
uint8_t Pubkey_B[2*ECCP_MAX_BYTE_LEN] = {0};
uint8_t ComputeKey_B[ECCP_MAX_BYTE_LEN] = {0};
uint32_t KeyByteLen = GET_BYTE_LEN(curve->eccp_p_bitLen);
```

1) ECCP_MAX_BYTE_LEN 宏表示 ECC 支持的最大曲线参数

2) 公钥由 x、y 坐标组成，因此长度为私钥的 2 倍

3) 密钥长度取决于使用的 ECC 曲线

ECDH 密钥协商：

```
/* get key pair for A */
ret = eccp_getkey((eccp_curve_t *)curve, Prikey_A, Pubkey_A);
if(PKE_SUCCESS != ret) {
    return PKE_ERROR;
}
/* get key pair for B */
ret = eccp_getkey((eccp_curve_t *)curve, Prikey_B, Pubkey_B);
if(PKE_SUCCESS != ret) {
    return PKE_ERROR;
}

/* key negotiate for A */
ret = ecdh_compute_key((eccp_curve_t *)curve, Prikey_A, Pubkey_B, ComputeKey_A, KeyByteLen, NULL);
if(ECDH_SUCCESS != ret) {
    return PKE_ERROR;
}
/* key negotiate for B */
ret = ecdh_compute_key((eccp_curve_t *)curve, Prikey_B, Pubkey_A, ComputeKey_B, KeyByteLen, NULL);
if(ECDH_SUCCESS != ret) {
    return PKE_ERROR;
}

/* compare key */
if(memcmp(ComputeKey_A, ComputeKey_B, KeyByteLen)) {
```

```
        return PKE_ERROR;
    }
```

1) 生成私钥 A 和公钥 A

2) 生成私钥 B 和公钥 B

3) 使用私钥 A 和公钥 B 进行密钥协商

4) 使用私钥 B 和公钥 A 进行密钥协商

5) 对比协商出的密钥

3.5.2.3 Openssl 交叉验证

使用 Openssl 生成的公钥需要转换成 CryptoPKE 算法库使用的公钥(x、y)坐标形式，转换方式根据使用的 Openssl 编程语言有所不同需要用户自行实现。

注：

1. 由于 *CryptoPKE* 算法库内部默认已经采用字节大端方式，因此和 *PC* 端参数交换不必再做大小端转换处理。
2. *Openssl* 生成的公钥必须转换成 *x*、*y* 坐标的十六进制数据形式，且按照前一半数据为 *x* 坐标，后一半数据为 *y* 坐标顺序存放。

3.5.3 ECIES

ECIES（Elliptic Curve Integrated Encryption Scheme）即椭圆曲线集成加密方案，它是一种混合的公钥加密方案，内部用到了椭圆曲线密钥交换（ECDH）、密钥派生函数（KDF），对称加密和消息验证码（MAC）。

规定 ECIES 流程主要的加密标准有 SECG SEC-1、ISO/IEC 18033-2、IEEE 1363a 和 ANSI X9.63。*CryptoPKE* 算法库所采用的标准是 ANSI X9.63。在该标准中定义了基于哈希算法的 KDF 函数（即 ANSI-X9.63-KDF），规定了采用的对称加密算法为异或加密，MAC 算法为 HMAC。ECIES 输出的密文由发送方的临时公钥 C1、被加密的消息 C2、消息验证码 C3 三部分组成。这里的 ECIES 使用的椭圆曲线方程为$y^2 = x^3 + ax + b$，支持的这类曲线如 secp192r1、secp224r1、secp256r1、secp384r1、secp521r1、brainpoolP160r1 、 brainpoolP192r1 、 brainpoolP256r1 、 brainpoolP320r1 、 brainpoolP384r1 、brainpoolP512r1 等。

CryptoPKE 算法库支持 ECIES 的密钥对生成，加密和解密。

与发送方临时公钥表示有关的说明：

ECIES 加密输出的结果由三部分组成：发送方的临时公钥 C1、被加密的消息 C2、消息验证码 C3。其中 C1 由标识 PC 和曲线上一个二维点(x, y)组成。若 C1 使用非压缩表示，则其表示为：0x04||x||y。

3.5.3.1 接口说明

表 3-4 ECIES 函数接口

函数名	描述
uint32_t eccp_get_pubkey_from_prikey (eccp_curve_t *curve, uint8_t *priKey, uint8_t *pubKey)	从私钥获得公钥
uint32_t eccp_getkey (eccp_curve_t *curve, uint8_t *priKey, uint8_t *pubKey)	生成随机密钥对
uint32_t ansi_x963_2001_ecies_encrypt (eccp_curve_t *curve, uint8_t *msg, uint32_t msg_bytes, uint8_t *shared_info1, uint32_t shared_info1_bytes, uint8_t *shared_info2, uint32_t shared_info2_bytes, uint8_t *sender_tmp_pri_key, uint8_t *receiver_pub_key, EC_POINT_FORM point_form, HASH_ALG kdf_hash_alg, HASH_ALG mac_hash_alg, uint32_t mac_k_bytes, uint8_t *cipher, uint32_t *cipher_bytes)	使用 ECIES 加密明文
uint32_t ansi_x963_2001_ecies_decrypt(eccp_curve_t *curve, uint8_t *cipher, uint32_t cipher_bytes, uint8_t *receiver_pri_key, uint8_t *shared_info1, uint32_t shared_info1_bytes, uint8_t *shared_info2, uint32_t shared_info2_bytes, HASH_ALG kdf_hash_alg, HASH_ALG mac_hash_alg, uint32_t mac_k_bytes, uint8_t *msg, uint32_t *msg_bytes)	使用 ECIES 对密文进行解密

1) **eccp_get_pubkey_from_prikey**

函数名	eccp_get_pubkey_from_prikey
函数原型	uint32_t eccp_get_pubkey_from_prikey (eccp_curve_t *curve, uint8_t *priKey, uint8_t *pubKey)
功能	由私钥生成椭圆曲线公钥
输入	eccp_curve_t *curve 椭圆曲线结构体 eccp_curve_t 指针 uint8_t *priKey 私钥，字节大端表示
输出	uint8_t *pubKey 公钥，按顺序分别是 x 坐标和 y 坐标，字节大端表示
返回值	PKE_SUCCESS 成功 Other 失败，详见错误码定义
注意事项	1. 该接口声明在 pke.h 中，可用于 ECDSA 和 ECDH 的从私钥生成公钥。 2. 大数占用的字节数为曲线参数 p 或 n 占用的字节数，例如，若曲线是 secp521r1，那么私钥占用是 (521+7)/8=66 字节，公钥占用是 2*66=132 字节。

2) **eccp_getkey**

函数名	eccp_getkey
函数原型	uint32_t eccp_getkey (eccp_curve_t *curve, uint8_t *priKey, uint8_t *pubKey)
功能	椭圆曲线随机密钥对生成
输入	eccp_curve_t *curve 椭圆曲线结构体 eccp_curve_t 指针
输出	uint8_t *priKey 私钥，字节大端表示 uint8_t *pubKey 公钥，按顺序分别是 x 坐标和 y 坐标，字节大端表示
返回值	PKE_SUCCESS 成功 Other 失败，详见错误码定义
注意事项	1. 该接口声明在 pke.h 中，可用于 ECDSA 和 ECDH 的从私钥生成公钥。 2. 大数占用的字节数为曲线参数 p 或 n 占用的字节数，例如，若曲线是 secp521r1，那么私钥占用是 (521+7)/8=66 字节，公钥占用是 2*66=132 字节。

3) ansi_x963_2001_ecies_encrypt

函数名	ansi_x963_2001_ecies_encrypt
函数原型	uint32_t ansi_x963_2001_ecies_encrypt(eccp_curve_t *curve, uint8_t *msg, uint32_t msg_bytes, uint8_t *shared_info1, uint32_t shared_info1_bytes, uint8_t *shared_info2, uint32_t shared_info2_bytes, uint8_t *sender_tmp_pri_key, uint8_t *receiver_pub_key, EC_POINT_FORM point_form, HASH_ALG kdf_hash_alg, HASH_ALG mac_hash_alg, uint32_t mac_k_bytes, uint8_t *cipher, uint32_t *cipher_bytes);
功能	使用 ECIES 加密明文
输入	eccp_curve_t *curve　椭圆曲线结构体 eccp_curve_t 指针 uint8_t *msg　需要加密的原始消息 uint32_t msg_bytes　原始消息的字节长度 uint8_t *shared_info1　共享消息 1 用于 kdf 函数 uint32_t shared_info1_bytes　共享消息 1 字节长度 uint8_t *shared_info2　共享消息 2 用于 hmac 函数 uint32_t shared_info2_bytes　共享消息 2 的字节长度 uint8_t *sender_tmp_pri_key　发送方的临时私钥 uint8_t *receiver_pub_key　接收方的公钥 EC_POINT_FORM point_form　曲线上点的表示方法，详细取值查看注意事项 4 HASH_ALG kdf_hash_alg　kdf 函数所采用的哈希函数 HASH_ALG mac_hash_alg　hmac 函数所采用的的哈希函数 uint32_t mac_k_bytes　hmac 函数所使用的密钥长度
输出	uint8_t *cipher　密文 uint32_t *cipher_bytes　密文的字节长度
返回值	ECIES_SUCCESS　成功 Other　失败，详见错误码定义
注意事项	1. 如果不提供共享消息 1 (shared_info1)，请设置共享消息 1(shared_info1)为 NULL，共享消息 1(字节长度 shared_info1_bytes 为 0)。 2. 如果不提供共享消息 2(shared_info2)，请设置共享消息 2(shared_info2)为 NULL，共享消息 2(字节长度 shared_info2_bytes 为 0)。 3. 如果不提供发送方的临时私钥 sender_tmp_pri_key，请设置该项为 NULL，内部会产生发送方的临时私钥。 4. 对于曲线上点的表示方法 point_form，详细内容请参考本章第 3 小节，其取值为： #define POINT_COMPRESSED (0x02) #define POINT_UNCOMPRESSED (0x04) 5. hmac 的密钥长度 mac_k_bytes 不要超过下述宏定义： #define ECIES_MAC_KEY_MAX_BYTE_LEN (128)

4) **ansi_x963_2001_ecies_decrypt**

函数名	ansi_x963_2001_ecies_decrypt
函数原型	uint32_t ansi_x963_2001_ecies_decrypt(eccp_curve_t *curve, uint8_t *cipher, uint32_t cipher_bytes, uint8_t *receiver_pri_key, uint8_t *shared_info1, uint32_t shared_info1_bytes, uint8_t *shared_info2, uint32_t shared_info2_bytes, HASH_ALG kdf_hash_alg, HASH_ALG mac_hash_alg, uint32_t mac_k_bytes, uint8_t *msg, uint32_t *msg_bytes);
功能	使用 ECIES 解密得到明文
输入	eccp_curve_t *curve 椭圆曲线结构体 eccp_curve_t 指针 uint8_t *cipher 密文 uint32_t cipher_bytes 密文的字节长度 uint8_t *receiver_pri_key 接受者的私钥 uint8_t *shared_info1 共享消息 1 用于 kdf 函数 uint32_t shared_info1_bytes 共享消息 1 字节长度 uint8_t *shared_info2 共享消息 2 用于 hmac 函数 uint32_t shared_info2_bytes 共享消息 2 的字节长度 HASH_ALG kdf_hash_alg kdf 函数所采用的哈希函数 HASH_ALG mac_hash_alg hmac 函数所采用的哈希函数 uint32_t mac_k_bytes hmac 函数所使用的密钥长度
输出	uint8_t *msg 解密得到的原始消息 uint32_t *msg_bytes 原始消息的字节长度
返回值	ECIES_SUCCESS 成功 Other 失败，详见错误码定义
注意事项	1. 如果不提供共享消息 1 (shared_info1)，请设置共享消息 1(shared_info1)为 NULL，共享消息 1(字节长度 shared_info1_bytes 为 0)。 2. 如果不提供共享消息 2(shared_info2)，请设置共享消息 2(shared_info2)为 NULL，共享消息 2(字节长度 shared_info2_bytes 为 0)。 3. hmac 的密钥长度 mac_k_bytes 不要超过下述宏定义： #define ECIES_MAC_KEY_MAX_BYTE_LEN (128)

3.5.3.2 例程介绍

本文主要以 pke_ecies（applications/ crypto/ pke_ ecies）样例进行介绍，该样例展示了 PKE 外设使用 CryptoPKE 算法库进行 ECIES 算法的加密、解密。

1) 基本流程

ECIES 算法加密、解密流程如下图 3-5 所示：

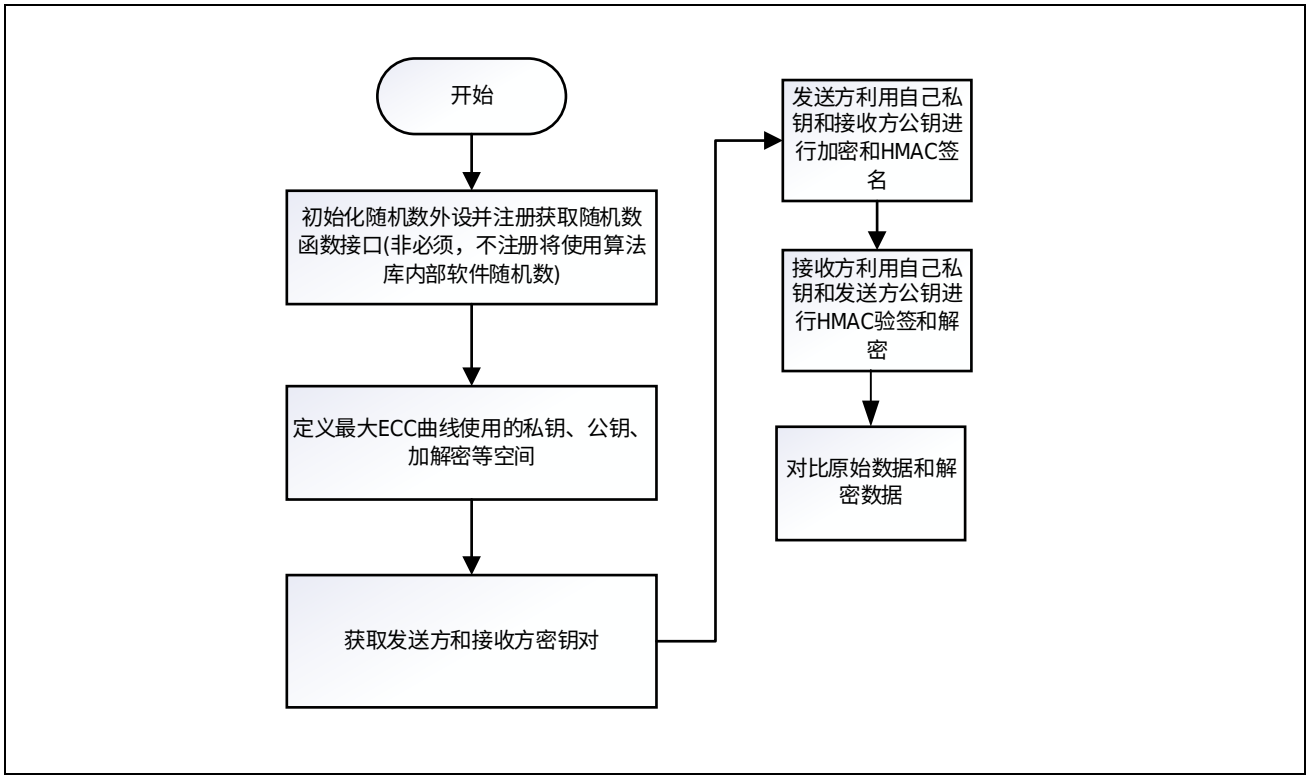

图 3-5 ECIES 加密、解密基本流程

2) 源码说明

随机数初始化与注册：

参考 RSA 随机数初始化与注册说明部分。

算法参数定义：

```
#define ECIES_MSG_MAX_LEN                (256U)
```

ECIES_MSG_MAX_LEN 定义用于加密数据的最大长度，加密数据由发送方的临时公钥 C1、被加密的消息 C2、消息验证码 C3 三部分组成。

算法参数定义:

```
uint8_t sender_pri_key[ECCP_MAX_BYTE_LEN];
uint8_t sender_pub_key[2 * ECCP_MAX_BYTE_LEN];
uint8_t receiver_pri_key[ECCP_MAX_BYTE_LEN];
uint8_t receiver_pub_key[2 * ECCP_MAX_BYTE_LEN];
uint8_t cipher[ECIES_MSG_MAX_LEN];
uint32_t cipher_bytes;
uint8_t replain[ECIES_MSG_MAX_LEN];
uint32_t replain_bytes;
uint32_t mac_key_bytes = ECIES_MAC_KEY_MAX_BYTE_LEN;
EC_POINT_FORM point_form = POINT_UNCOMPRESSED;
HASH_ALG kdf_hash_alg = HASH_SHA256;
HASH_ALG mac_hash_alg = HASH_SHA256;
```

1) ECCP_MAX_BYTE_LEN 宏表示 ECC 支持的最大曲线参数

2) 公钥由 x、y 坐标组成，因此长度为私钥的 2 倍

3) 使用非压缩方式

4) Kdf 和 hmac 采用 HASH_SHA256 算法

ECIES 加密、解密：

```
    /* Get sender key pair */
    ret = eccp_getkey((eccp_curve_t *)curve, sender_pri_key, sender_pub_key);
    if (ECIES_SUCCESS != ret) {
        return PKE_ERROR;
    }
    /* Get receiver key pair */
    ret = eccp_getkey((eccp_curve_t *)curve, receiver_pri_key, receiver_pub_key);
    if (ECIES_SUCCESS != ret) {
        return PKE_ERROR;
    }

    /* encrypt */
    ret = ansi_x963_2001_ecies_encrypt((eccp_curve_t *)curve, (uint8_t *)msg, strlen(msg), NULL, 0, NULL, 0,
sender_pri_key, receiver_pub_key, \
                        point_form, kdf_hash_alg, mac_hash_alg, mac_key_bytes, cipher, &cipher_bytes);
    if (ECIES_SUCCESS != ret) {
        return PKE_ERROR;
    }

    /* decrypt */
    ret = ansi_x963_2001_ecies_decrypt((eccp_curve_t *)curve, cipher, cipher_bytes, receiver_pri_key, NULL, 0, NULL,
0, \
                            kdf_hash_alg, mac_hash_alg, mac_key_bytes, replain, &replain_bytes);
    if (ECIES_SUCCESS != ret) {
        return PKE_ERROR;
    }

    /* compare data */
    if (memcmp((uint8_t *)msg, replain, replain_bytes)) {
        return PKE_ERROR;
    }
```

1) 生成发送方和接收方密钥对

2) 发送方使用自己私钥和接收方公钥进行加密和 HMAC 签名

3) 接收方使用自己私钥和发送方公钥进行 HAMC 验签和解密

4) 对比数据

3.6 SM2

3.6.1 概述

SM2 算法是国密局颁发的一套椭圆曲线公钥密码算法。具体的，可分签名验签算法、加解密算法、密钥协商（或密钥交换）算法三个子类。

SM2 采用素数域 GF(p)上的椭圆曲线$y^2 = x^3 + ax + b$，这里使用推荐曲线，如下是曲线参数：

p=0xFFFFFFFEFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF00000000FFFFFFFFFFFFFFFF

a=0xFFFFFFFEFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF00000000FFFFFFFFFFFFFFFC

b=0x28E9FA9E9D9F5E344D5A9E4BCF6509A7F39789F515AB8F92DDBCBD414D940E93

n=0xFFFFFFFEFFFFFFFFFFFFFFFFFFFFFFFF7203DF6B21C6052B53BBF40939D54123

Gx =0x32C4AE2C1F1981195F9904466A39C9948FE30BBFF2660BE1715A4589334C74C7

Gy =0xBC3736A2F4F6779C59BDCEE36B692153D0A9877CC62A474002DF32E52139F0A0

点(G_x, G_y)是曲线的基点，其阶是素数 n，n 同时也是曲线的阶（即曲线上点的个数）。

CryptoPKE 算法库支持 SM2 算法的所有功能。

SM2 密文有三部分，即 C1、C2、C3，但是顺序表示有过两种建议，顺序 C1C3C2 是当前的建议，C1C2C3 则是较早之前的，用户可选择顺序类型，或自行进行顺序调整。

3.6.2 接口说明

表 3-5 SM2 函数接口

函数名	描述
uint32_t sm2_getZ (uint8_t *ID, uint32_t byteLenofID, uint8_t pubKey[65], uint8_t Z[32])	获取 Z 值
uint32_t sm2_getE (uint8_t *M, uint32_t byteLen, uint8_t Z[32], uint8_t [32])	获取 E 值
uint32_t sm2_get_pubkey_from_prikey (uint8_t priKey[32], uint8_t pubKey[65])	由私钥生成 SM2 公钥
uint32_t sm2_getkey (uint8_t priKey[32], uint8_t pubKey[65])	SM2 随机密钥对生成
uint32_t sm2_sign (uint8_t E[32], uint8_t rand_k[32], uint8_t priKey[32], uint8_t signature[64])	SM2 签名生成
uint32_t sm2_verify(uint8_t E[32], uint8_t pubKey[65], uint8_t signature[64])	SM2 签名验证
uint32_t sm2_encrypt(uint8_t *M, uint32_t MByteLen, uint8_t rand_k[32], uint8_t pubKey[65], sm2_cipher_order_e order, uint8_t *C, uint32_t *CByteLen)	SM2 消息加密
uint32_t sm2_decrypt(uint8_t *C, uint32_t CByteLen, uint8_t priKey[32], sm2_cipher_order_e order, uint8_t *M, uint32_t *MByteLen)	SM2 消息解密
uint32_t sm2_exchangekey(sm2_exchange_role_e role, uint8_t *dA, uint8_t *PB, uint8_t *rA, uint8_t *RA, uint8_t *RB, uint8_t *ZA, uint8_t *ZB, uint32_t kByteLen, uint8_t *KA, uint8_t *S1, uint8_t *SA)	SM2 密钥协商

1) **sm2_getZ**

函数名	sm2_getZ
函数原型	uint32_t sm2_getZ (uint8_t *ID, uint32_t byteLenofID, uint8_t pubKey[65],

	uint8_t Z[32])
功能	SM2 获取 Z 值
输入	uint8_t *ID 用户 ID uint32_t byteLenofID 用户 ID 的字节长度，要小于 8192 uint8_t pubKey[65] 用户的公钥
输出	uint8_t Z[32] Z 值，32 字节
返回值	SM2_SUCCESS 成功 Other 失败，详见错误码定义
注意事项	1. Z 值用在 SM2 签名获取 E 值，以及密钥协商中。 2. 根据《SM2 密码算法使用规范》，无特殊约定的情况下，用户标识 ID 的长度为 16 字节，其默认值从左至右依次如下： 0x31,0x32,0x33,0x34,0x35,0x36,0x37,0x38,0x31,0x32,0x33,0x34,0x35,0x36,0x37,0x38 即没有用到用户标识的情况下，请使用该默认 ID。实际上，若 ID 为 NULL 或者为 0，则该接口会自动使用该默认 ID。 3. 该接口实际是调用 SM3 接口计算 hash。

2) **sm2_getE**

函数名	sm2_getE
函数原型	uint32_t sm2_getE (uint8_t *M, uint32_t byteLen, uint8_t Z[32], uint8_t E[32])
功能	SM2 获取 E 值
输入	uint8_t *M 消息 M uint32_t byteLen 消息 M 的字节长度 uint8_t Z[32] Z 值，32 字节
输出	uint8_t E[32] 计算得到的 E 值，32 字节
返回值	SM2_SUCCESS 成功 Other 失败，详见错误码定义
注意事项	1. E 值用在 SM2 签名中。 2. 该接口实际是调用 SM3 接口计算 hash。

3) **sm2_get_pubkey_from_prikey**

函数名	sm2_get_pubkey_from_prikey
函数原型	uint32_t sm2_get_pubkey_from_prikey (uint8_t priKey[32], uint8_t pubKey[65])
功能	由私钥生成 SM2 公钥
输入	uint8_t priKey[32] SM2 私钥，32 字节
输出	uint8_t pubKey[65] SM2 公钥，65 字节
返回值	SM2_SUCCESS 成功 Other 失败，详见错误码定义
注意事项	1. 私钥是 256bit，即 32 字节的大数，公钥是对应的一个点，65 字节，其首字节是 0x04，表示该点的 x 和 y 坐标直接存储表示没有被压缩；即首字节后是 32 字节的 x 坐标和 32 字节的 y 坐标。

4) **sm2_getkey**

函数名	sm2_getkey
函数原型	uint32_t sm2_getkey (uint8_t priKey[32], uint8_t pubKey[65])
功能	SM2 随机密钥对生成
输入	无
输出	uint8_t priKey[32] SM2 私钥，32 字节 uint8_t pubKey[65] SM2 公钥，65 字节
返回值	SM2_SUCCESS 成功 Other 失败，详见错误码定义
注意事项	1. 私钥是 256bit，即 32 字节的大数，公钥是对应的一个点，65 字节，其首字节是 0x04，表示该点的 x 和 y 坐标直接存储表示没有被压缩；即首字节后是 32 字节的 x 坐标和 32 字节的 y 坐标。

5) **sm2_sign**

函数名	sm2_sign
函数原型	uint32_t sm2_sign (uint8_t E[32], uint8_t rand_k[32], uint8_t priKey[32], uint8_t signature[64])
功能	SM2 签名生成
输入	uint8_t E[32] 待签名的 E 值，32 字节 uint8_t rand_k[32] 签名中的实时随机大整数 k，32 字节；若没有该参数，请设置该参数为 NULL 即可，函数内部会自动生成，这也是推荐的做法 uint8_t priKey[32] 签名者的私钥，32 字节
输出	uint8_t signature[64] 签名结果，64 字节
返回值	SM2_SUCCESS 成功 Other 失败，详见错误码定义
注意事项	1. 对消息进行签名，必须先调用 sm2_getZ()获得 Z 值，再调用 sm2_getE()获得 E 值。最终调用本接口得到签名结果。 2. 若没有 rand_k，请将该参数设置为 NULL，函数内部会实时生成一个随机 rand_k 参与签名计算，这也是推荐的做法。 3. 签名结果 64 字节，依次是 32 字节的大数 r 和 32 字节的大数 s。

6) **sm2_verify**

函数名	sm2_verify
函数原型	uint32_t sm2_verify (uint8_t E[32], uint8_t pubKey[65], uint8_t signature[64])
功能	SM2 签名验证
输入	uint8_t E[32] 待签名的 E 值，32 字节 uint8_t pubKey[65] 签名验证用的公钥，65 字节 uint8_t signature[64] 签名结果，64 字节
输出	无

返回值	SM2_SUCCESS 成功 Other 失败，详见错误码定义
注意事项	1. 对消息进行签名验证，须先调用 sm2_getZ 获得 Z 值，再调用 sm2_getE 获得 E 值。最终调用本接口得到验证结果。

7) sm2_encrypt

函数名	sm2_encrypt
函数原型	uint32_t sm2_encrypt (uint8_t *M, uint32_t MByteLen, uint8_t rand_k[32], uint8_t pubKey[65], sm2_cipher_order_e order, uint8_t *C, uint32_t *CByteLen)
功能	SM2 加密
输入	uint8_t *M 待加密的明文，MByteLen 字节 uint32_t MByteLen 待加密的明文的字节长度，必须大于 0 uint8_t rand_k[32] 加密中的实时随机大整数 k，32 字节；若没有该参数，请设置该参数为 NULL 即可，函数内部会自动生成，这也是推荐的做法 uint8_t pubKey[65] 加密用的公钥，65 字节 sm2_cipher_order_e order 密文顺序标识
输出	uint8_t *C 加密得到的密文，注意其指向空间要保证至少是 MByteLen+97 字节 uint32_t *CByteLen 密文长度，正常情况下是 MByteLen+97 字节
返回值	SM2_SUCCESS 成功 Other 失败，详见错误码定义
注意事项	1. 密文分三部分，C1 是曲线上一个点，65 字节，其结构和公钥一致；C2 和明文一样长；C3 是保证完整性的校验值，为 SM3 算法结果，32 字节。 2. 若没有 rand_k，请将该参数设置为 NULL，函数内部会实时生成一个随机 rand_k 参与签名计算，这也是推荐的做法。 3. 若明文长度很短，如 1 字节，则会有 1/256 的概率加密失败（返回 SM2_ZERO_ALL），因为此时内部 kdf 结果也是 1 个字节，当它为全 0 时则返回 SM2_ZERO_ALL，这是正常的。此时可以重新尝试加密（rand_k 设置为 NULL），或者更换 rand_k 值直至返回成功即可。 4. 为节省空间，可以让 M 和 C 指向同一块 buffer，调用后原输入 M 会被密文 C 覆盖。 5. 若 M 和 C 不指向同一个 buffer，但两块 buffer 占用有交叉，则会返回失败，此时请保证各自 buffer 独立不交叉。

8) sm2_decrypt

函数名	sm2_decrypt
函数原型	uint32_t sm2_decrypt (uint8_t *C, uint32_t CByteLen, uint8_t priKey[32], sm2_cipher_order_e order, uint8_t *M, uint32_t *MByteLen)
功能	SM2 解密

输入	uint8_t *C 密文，CByteLen 字节 uint32_t CByteLen 密文长度，必须大于 97 字节 uint8_t priKey[32] 解密用到的私钥 sm2_cipher_order_e order 密文顺序标识
输出	uint8_t *M 解密得到的明文，注意其指向空间要保证至少是 CbyteLen-97 字节 uint32_t *MByteLen 明文长度，正常情况下是 CbyteLen-97 字节
返回值	SM2_SUCCESS 成功 Other 失败，详见错误码定义
注意事项	1. 密文分三部分，C1 是曲线上一个点，65 字节，其结构和公钥一致；C2 和明文一样长；C3 是保证完整性的校验值，为 SM3 算法结果，32 字节。 2. 输入的密文 C 和其顺序标识 order 必须相符，否则解密失败。 3. 为节省空间，可以让 M 和 C 指向同一块 buffer，调用后原输入 C 会被明文 M 覆盖。 4. 若 M 和 C 不指向同一个 buffer，但两块 buffer 占用有交叉，则会返回失败，此时请保证各自 buffer 独立不交叉。

9) **sm2_exchangekey**

函数名	sm2_exchangekey
函数原型	uint32_t sm2_exchangekey (sm2_exchange_role_e role, uint8_t *dA, uint8_t *PB, uint8_t *rA, uint8_t *RA, uint8_t *RB, uint8_t *ZA, uint8_t *ZB, uint32_t kByteLen, uint8_t *KA, uint8_t *S1, uint8_t *SA)
功能	SM2 密钥协商
输入	sm2_exchange_role_e role 己方用户角色，只能是 SM2_Role_Sponsor 或 SM2_Role_Responsor uint8_t *dA 己方私钥，32 字节 uint8_t *PB 对方公钥，65 字节 uint8_t *rA 己方临时私钥，32 字节 uint8_t *RA 己方临时公钥，65 字节 uint8_t *RB 对方临时公钥，65 字节 uint8_t *ZA 己方 Z 值，32 字节 uint8_t *ZB 对方 Z 值，32 字节 uint32_t kByteLen 协商密钥的字节长度
输出	uint8_t *KA 协商出的密钥，kByteLen 字节 uint8_t *S1 可选，密钥交换发起者的 S1，或者密钥交换响应者的 S2，32 字节 uint8_t *SA 可选，密钥交换发起者的 SA，或者密钥交换响应者的 SB，32 字节
返回值	SM2_SUCCESS 成功 Other 失败，详见错误码定义
注意事项	1. 只有当参数 S1 和 SA 都不是 NULL 时，才会计算这两个输出，否则不会计算，即 S1 和 SA 是可选的。

	2. 若协商成功，则双方协商出来的 KA 相同。此时若还生成了 S1/S2 和 SA/SB，则发送方的 S1 和接收方的 SB 相同，发送方的 SA 和接收方的 S2 相同。

3.6.3 例程介绍

本文主要以 pke_sm2（applications/ crypto/ pke_sm2）样例进行介绍，该样例展示了 PKE 外设使用 CryptoPKE 算法库进行 SM2 算法的加密解密、签名验签、密钥协商。

1) 基本流程

SM2 算法加密、解密流程如下图 3-6 所示：

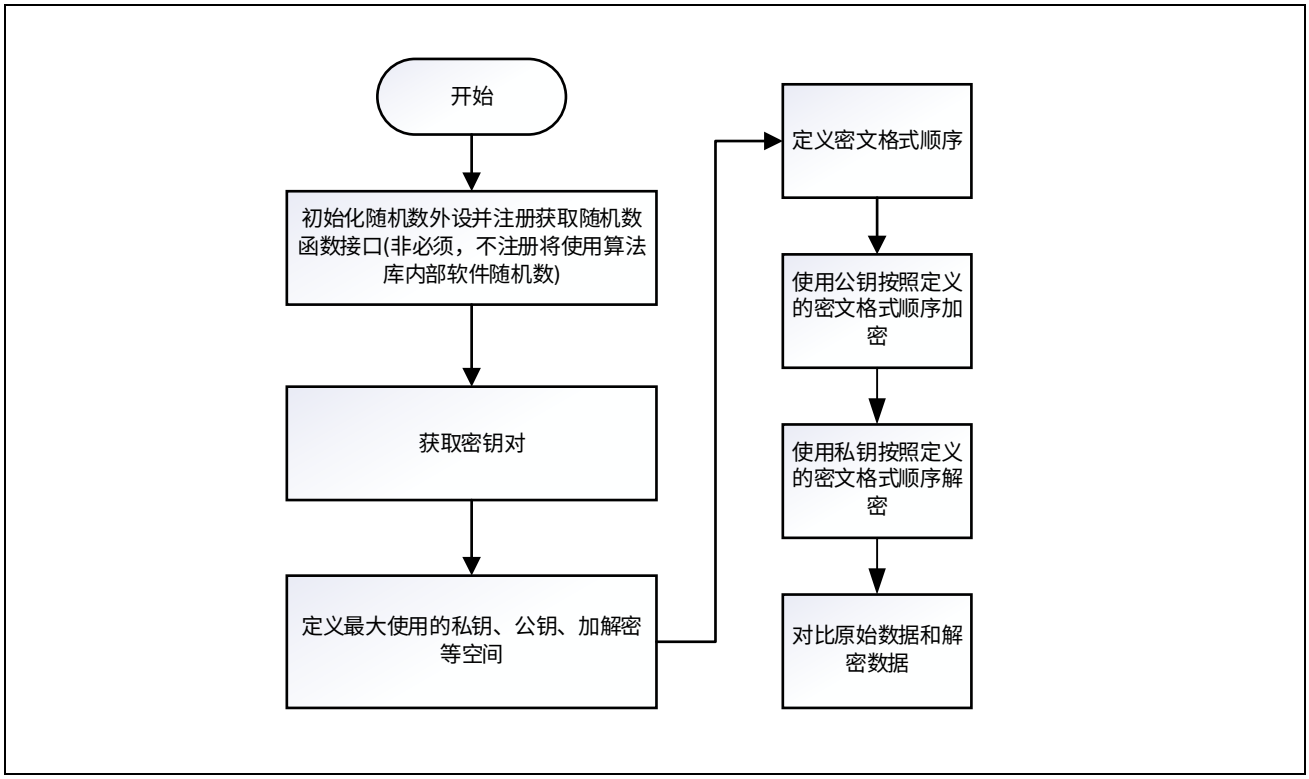

图 3-6 SM2 加密、解密基本流程

SM2 算法签名、验签流程如下图 3-7 所示：

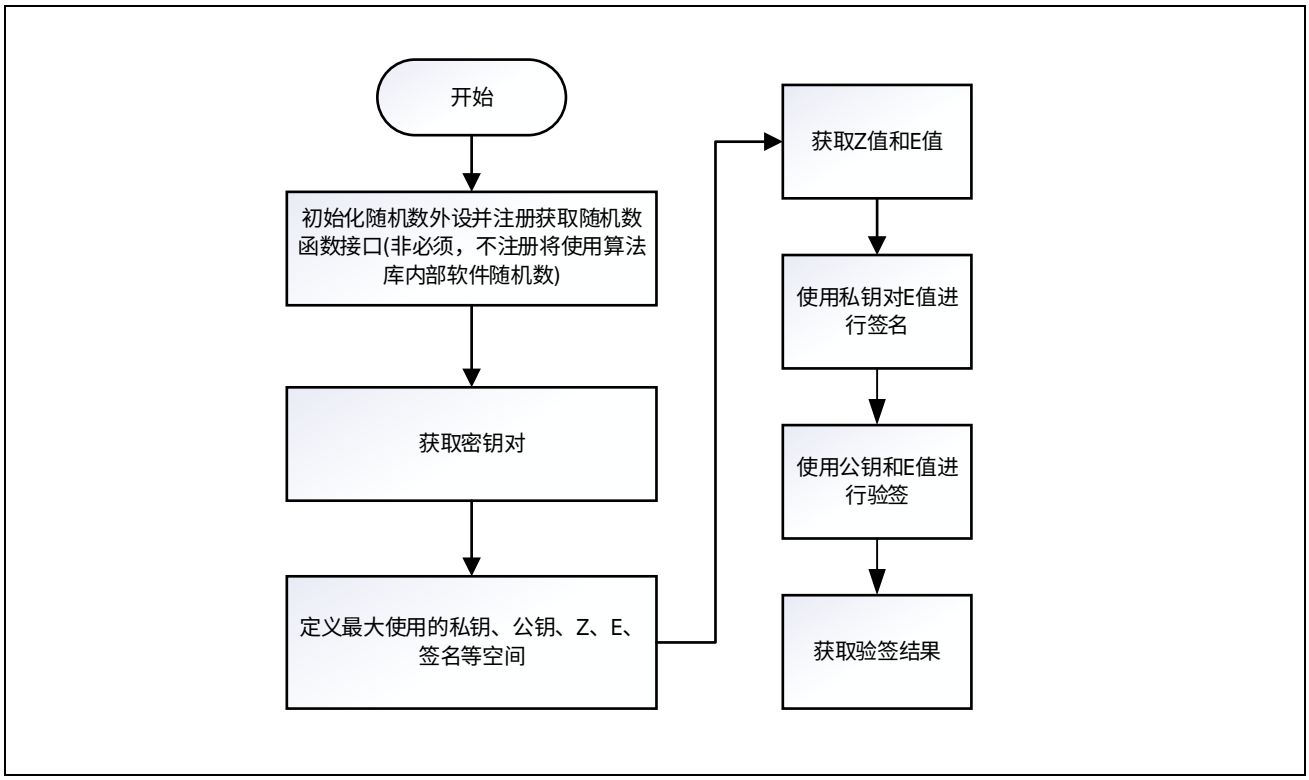

图 3-7 SM2 签名、验签基本流程

SM2 算法密钥交换流程如下图 3-8 所示：

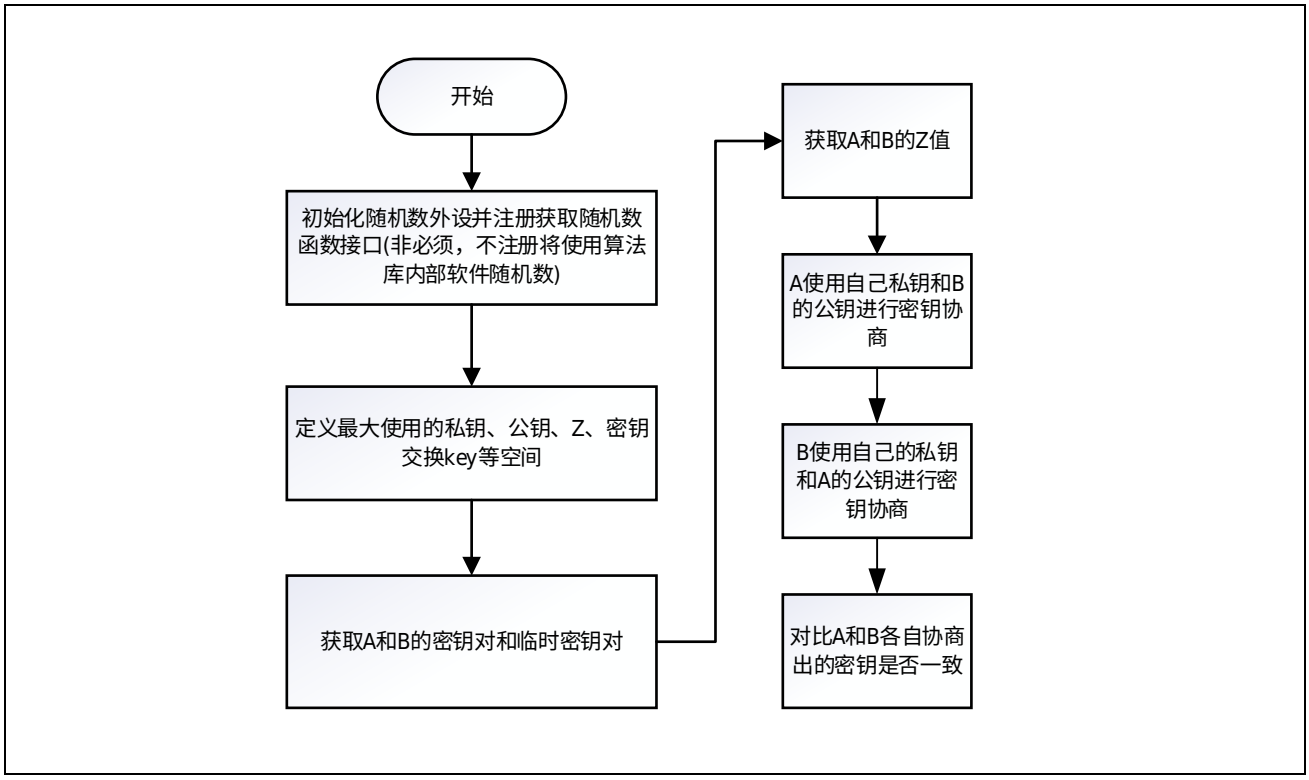

图 3-8 SM2 密钥交换基本流程

2) 源码说明

随机数初始化与注册：

参考 RSA 随机数初始化与注册说明部分。

算法参数定义：

```
#define PRIVATE_KEY_LEN         (32U)                       /* 256bit */
#define PUBLIC_KEY_LEN          ((PRIVATE_KEY_LEN * 2U) + 1U)   /* x y points and 0x04 head */
#define SM2_MSG_MAX_LEN         (256U)
```

1) PUBLIC_KEY_LEN 由 xy 坐标加头部的 0x04 组成

2) SM2_MSG_MAX_LEN 定义原始消息数据的最大长度

SM2 加密、解密：

算法参数定义:

```
uint8_t in[SM2_MSG_MAX_LEN] = {0};
uint8_t ciphertext[SM2_MSG_MAX_LEN+97] = {0};      /* C2(MSG) + C1(65byte) + C3(32byte) */
uint8_t plaintext[SM2_MSG_MAX_LEN] = {0};
sm2_cipher_order_e order = SM2_C1C3C2;
```

1) 加密数据由 C1C2C3 组成，C2 和明文一样长，C1 固定占用 65 字节，C3 固定占用 32 字节

2) 密文采用 C1C3C2 顺序格式

```
/* Raw data */
msg = "SM2 encrypt & decrypt test";
memcpy(in, msg, strlen(msg));

/* Encrypt */
ret = sm2_encrypt(in, strlen(msg), NULL, publickey, order, ciphertext, &sm2_out_bytelen);
if(SM2_SUCCESS != ret) {
    return PKE_ERROR;
}

/* Decrypt */
 ret = sm2_decrypt(ciphertext, sm2_out_bytelen, privatekey, order, plaintext, &out_bytelen);
if(SM2_SUCCESS != ret) {
    return PKE_ERROR;
}

/* Compare data */
if (memcmp(in, plaintextt, out_bytelen)) {
    return PKE_ERROR;
```

```
}
```

1) 使用公钥按照 order 定义密文顺序加密数据

2) 使用私钥按照 order 定义顺序解密数据

3) 比较原始数据和解密后的数据

SM2 签名、验签：

算法参数定义：

```
uint8_t Z[32], E[32];
uint8_t signature[64];
const char *msg = "SM2 sign & verify test";
const char *id = "1234567812345678";            /* default id */
```

1) 签名数据由 r、s 组成，各占用 32 字节

2) 默认 id 为[0x31,0x32,0x33,0x34,0x35,0x36,0x37,0x38,0x31,0x32,0x33,0x34,0x35,0x36,0x37,0x38]，等效字符串为”1234567812345678”

```
/* get Z */
ret = sm2_getZ((uint8_t *)id, strlen(id), publickey, Z);
if(SM2_SUCCESS != ret) {
    return PKE_ERROR;
}

/* get E */
ret = sm2_getE((uint8_t *)msg, strlen(msg), Z, E);
if(SM2_SUCCESS != ret) {
    return PKE_ERROR;
}

/* sign */
ret = sm2_sign(E, NULL, privatekey, signature);
if((SM2_SUCCESS != ret)) {
    return PKE_ERROR;
}

/* verify */
ret = sm2_verify(E, publickey, signature);
if(SM2_SUCCESS != ret) {
    return PKE_ERROR;
}
```

1) 使用 ID 和公钥获取 Z 值

2) 由 Z 值和原始数据获取 E 值

3) 使用私钥对 E 值进行签名

4) 使用公钥和 E 值验证签名

SM2 密钥协商：

算法参数定义：

```
uint8_t dA[PRIVATE_KEY_LEN], dB[PRIVATE_KEY_LEN];                /* A and B private key */
uint8_t PA[PUBLIC_KEY_LEN], PB[PUBLIC_KEY_LEN];                  /* A and B public key */
uint8_t rA[PRIVATE_KEY_LEN], rB[PRIVATE_KEY_LEN];                /* A and B temporary private key */
uint8_t RA[PUBLIC_KEY_LEN], RB[PUBLIC_KEY_LEN];                  /* A and B temporary public key */
uint8_t ZA[32], ZB[32];                                          /* The Z value of A and B */
uint8_t KA[EXCHANGE_KBYTE_LEN], KB[EXCHANGE_KBYTE_LEN];          /* The negotiated key of A and B    */
const char *id = "1234567812345678";                             /* default id */
```

1) 定义 A、B 各自公钥私钥和临时公钥私钥空间

2) 定义 A、B 各自 Z 空间

3) 定义 A、B 各自协商密钥空间

4) 默认 id 为[0x31,0x32,0x33,0x34,0x35,0x36,0x37,0x38,0x31,0x32,0x33,0x34,0x35,0x36,0x37,0x38]，等效字符串为”1234567812345678”

```
/* Get the key pair and temporary key pair of A */
ret = sm2_getkey(dA, PA);
ret |= sm2_getkey(rA, RA);
if(SM2_SUCCESS != ret) {
    return PKE_ERROR;
}

/* Get the key pair and temporary key pair of B */
ret = sm2_getkey(dB, PB);
ret |= sm2_getkey(rB, RB);
if(SM2_SUCCESS != ret) {
    return PKE_ERROR;
}

/* Get Z */
ret = sm2_getZ((uint8_t *)id, strlen(id), PA, ZA);
ret |= sm2_getZ((uint8_t *)id, strlen(id), PB, ZB);
if(SM2_SUCCESS != ret) {
    return PKE_ERROR;
```

```
}

/* A exchange */
ret = sm2_exchangekey(SM2_Role_Sponsor, dA, PB, rA, RA, RB, ZA, ZB, EXCHANGE_KBYTE_LEN, KA, NULL, NULL);
if(SM2_SUCCESS != ret) {
    return PKE_ERROR;
}

/* B exchange */
ret = sm2_exchangekey(SM2_Role_Responsor, dB, PA, rB, RB, RA, ZB, ZA, EXCHANGE_KBYTE_LEN, KB, NULL, NULL);
if(SM2_SUCCESS != ret) {
    return PKE_ERROR;
}

/* Compare key */
if(memcmp(KA, KB, EXCHANGE_KBYTE_LEN)) {
    return PKE_ERROR;
}
```

1) 获取 A、B 各自的密钥对和临时密钥对

2) 使用 A、B 各自 ID 和公钥获取 Z 值

3) A 使用私钥和 B 的公钥以及 Z 值进行密钥协商

4) B 使用私钥和 A 的公钥以及 Z 值进行密钥协商

5) 比较协商出的密钥

4 总结

本篇应用笔记详细介绍了 CRYPTO_PKE 算法库使用流程，用户须结合用户手册对其进行正确的配置并合理的应用。

版本修订记录

版本号	修订日期	修订内容
Rev1.00	2025/03/11	初版发布。
Rev1.01	2025/06/26	新增 HC32A4A8 系列型号，手册名称改为“AN_HC32F4A8_A4A8 系列 CRYPTO_PKE 算法库使用_Rev1.01”。