M4521 ECC认证

NuMicro® 32位系列微控制器范例代码介绍

文件信息

应用简述	于 M4521 开发板上透过 mbedtls 函数库以纯软件实现 ECC 加解密运算功能
BSP 版本	M4521_BSP_CMSIS_V3.00.003
开发平台	NuTiny-SDK-M4521S

The information described in this document is the exclusive intellectual property of Nuvoton Technology Corporation and shall not be reproduced without permission from Nuvoton.

Nuvoton is providing this document only for reference purposes of NuMicro microcontroller and microprocessor based system design. Nuvoton assumes no responsibility for errors or omissions.

All data and specifications are subject to change without notice.

For additional information or questions, please contact: Nuvoton Technology Corporation.

www.nuvoton.com

1. 概述

Mbedtls 是由 Mbed TLS 在 Apache 2.0 或 GPL 2.0 许可下以开源形式提供的 SSL(Secure Sockets Layer)函数库，含有丰富的加解密函数可用来进行安全通讯，避免数据遭窃听。

此范例程序利用 mbedtls 函数库，以纯软件执行椭圆曲线密码学(Elliptic Curve Cryptography, ECC)之密钥加解密运算，以缓解部分开发板由于设计规格不同，而无法以硬件实现加解密功能的问题。

1.1 椭圆曲线密码学

椭圆曲线密码学是基于椭圆曲线的数学原理，属对称加密技术的公钥加密体制。在同等的安全等级下，相较于其它加密算法，ECC 具有更短的密钥，也能在此基础上提供更高的安全性等级加密，例如 256 bit 的 ECC 密钥强度相当于 3072 bit 的 RSA 密钥。ECC 数学逻辑更加复杂，故具有加解密较其它方法费时的缺点，但普遍认为 ECC 是给定密钥长度情况下，最强大的非对称加密算法。

ECC 的数学基础是利用椭圆曲线上的离散对数问题，针对密码学应用的椭圆曲线是在有限域的平面曲线[1,2]，其中 ECC 最常使用的椭圆曲线方程式如下：

$$y^2 = x^3 + ax + b \quad (\mathrm{a,b} \in \mathrm{GF(p)}, 4a^3 + 27b^2 \neq 0)$$

其中 a 和 b 为任意实数，椭圆曲线会因应 a 和 b 的数值而在平面上呈不同形状，但椭圆曲线始终会对称于 x 轴，且方程式需满足无奇异点，即图形没有尖点或自相交[3]，随机椭圆曲线函数如图 1-1、图 1-2 所示。

1 https://zh.m.wikipedia.org/zh-tw/%E6%A4%AD%E5%9C%86%E6%9B%B2%E7%BA%BF%E5%AF%86%E7%A0%81%E5%AD%A6
2 https://zhuanlan.zhihu.com/p/42629724
3 https://zh.m.wikipedia.org/zh-tw/%E6%A4%AD%E5%9C%86%E6%9B%B2%E7%BA%BF

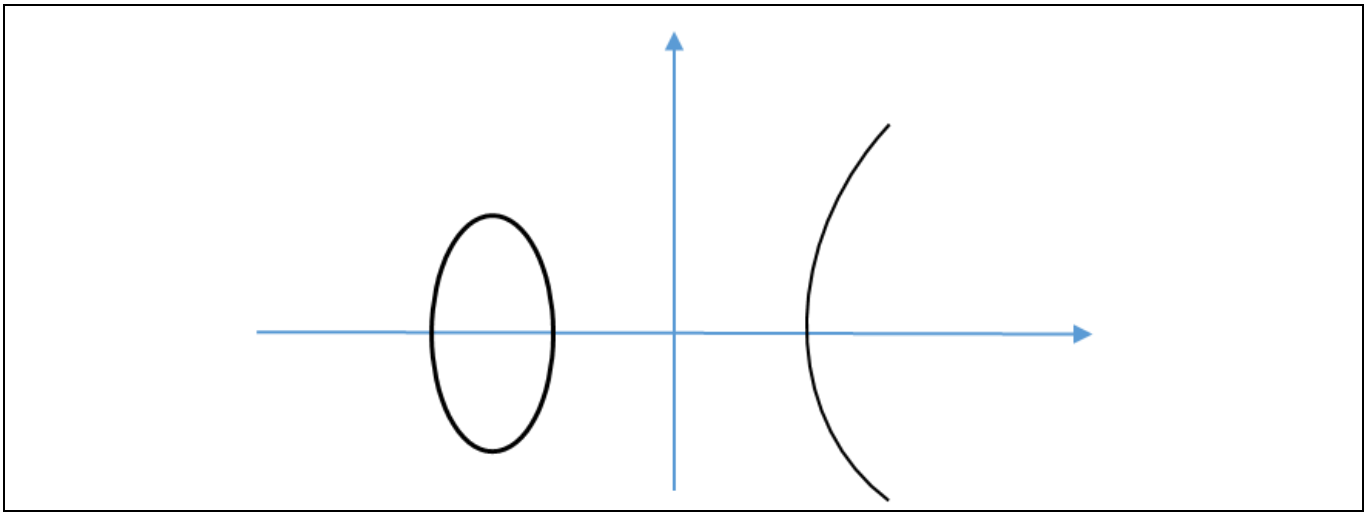

图 1-1 椭圆曲线图($y^2 = x^3 - 1$)

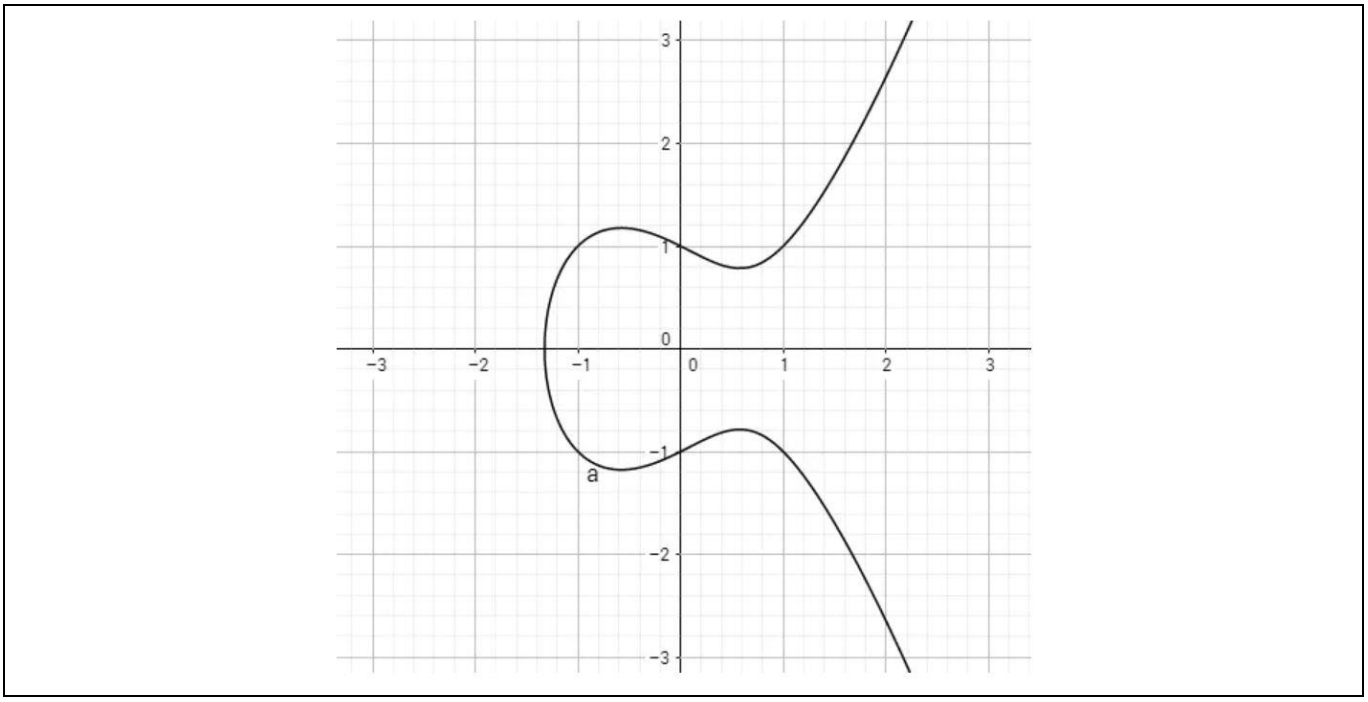

图 1-2 椭圆曲线图($y^2 = x^3 - x + 1$) [4]

4 https://www.jianshu.com/p/899077117ff8

1.2 椭圆曲线数字签名算法(ECDSA)

1.2.1 原理与操作

建立基于椭圆曲线的加密机制，需要找到类似 RSA 质因子分解或其他求离散对数这样的难题。而椭圆曲线上的已知 G 和 xG 求 x，是非常困难的，此即为椭圆曲线上的的离散对数问题，因此椭圆曲线数字签名(Elliptic Curve Digital Signature Algorithm, ECDAS)便由此而生。

在密码学中会生成一对密钥:公钥(public key)与私钥(private key)，相对椭圆曲线的加密机制，即在椭圆曲线上设 k 为私钥，k=kG 为公钥(G 为基点)，私钥负责加密，并由发讯者保管；公钥则对外公布，拥有者可用来解密对应数据。

当明文经签章算法产生摘要后，再以私钥加密形成数字签名才对外公布(非由私钥直接加密明文)，拥有对应公钥的使用者便能以公钥将签章解密，解出的摘要会与发送方加密前的内容一致，并藉此确认数据没被窜改过。ECDSA 加解密流程如图 1-3~图 1-4 所示。

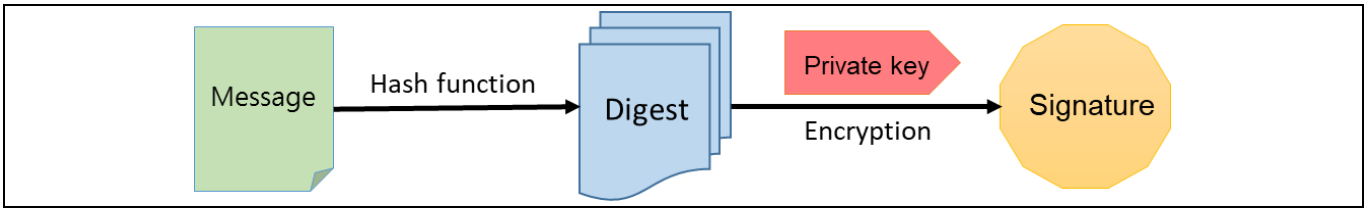

图 1-3 私钥加密形成签章

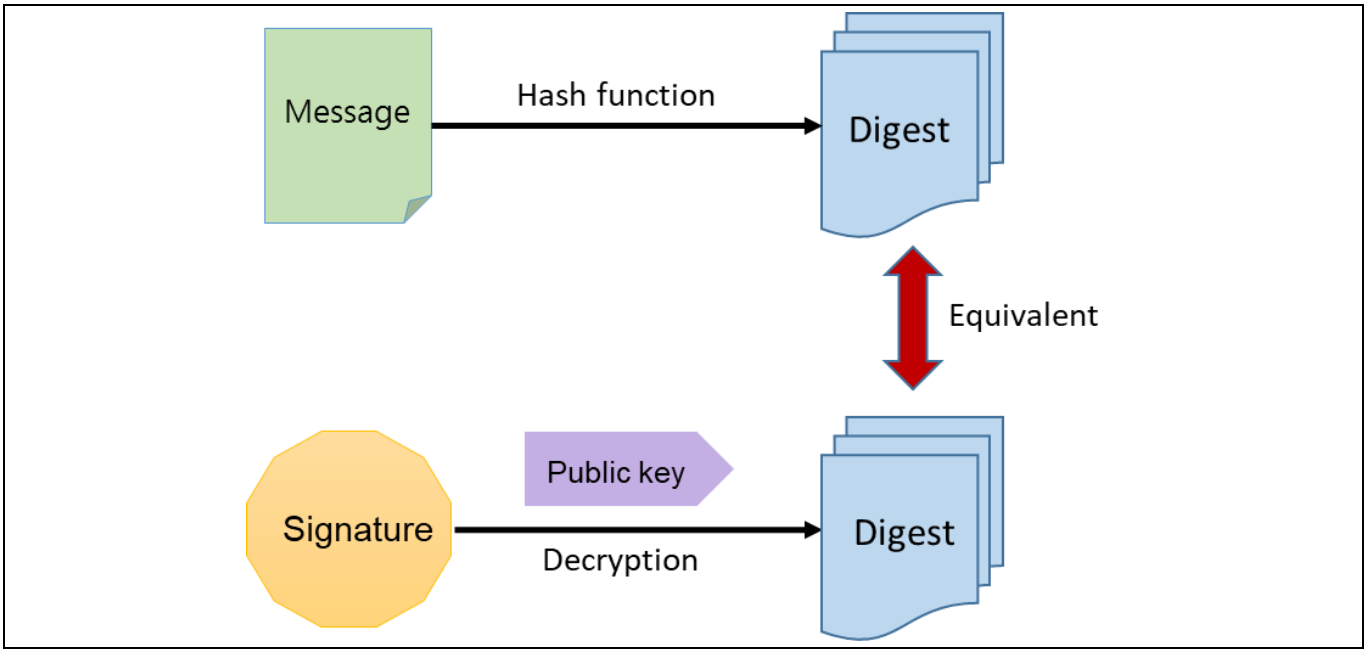

图 1-4 公钥对签章解密

1.2.2 密钥生成

首先依需求(位数、签章遵循标准等)选择一组椭圆曲线，并由 mbedtls 函数库调用 ***mbedtls_ecp_curve_info_from_grp_id()***取得该曲线相关信息，用处理过的参数 gid 投入函数 ***mbedtls_ecdsa_genkey()*** 生成椭圆曲线上的密钥对，最后再由 ***mbedtls_ecdsa_from_keypair()***基于密钥对产生 ECDSA 密钥参数。详细函数说明如表 1-1~表 1-3 所示。

*mbedtls_ecp_curve_info_from_grp_id(mbedtls_ecp_group_id grp_id)		
参数：	grp_id	椭圆曲线参数
返回值：		Null

表 1-1 椭圆曲线信息撷取 API

mbedtls_ecdsa_genkey(mbedtls_ecdsa_context *ctx, mbedtls_ecp_group_id gid, int (*f_rng)(void *, unsigned char *, size_t), void *p_rng)		
参数：	*ctx	ECDSA密钥参数
	gid	椭圆曲线参数
	*f_rng	随机数函数
	*p_rng	f_rng 附加函数 (不使用则填 NULL)
返回值：		成功 = 0 失败 = MBEDTLS_ERR_ECP_XXX

表 1-2 由椭圆曲线生成密钥对 API

mbedtls_ecdsa_from_keypair(mbedtls_ecdsa_context *ctx, const mbedtls_ecp_keypair *key)		
参数：	*ctx	ECDSA密钥参数
	*key	使用密钥对
返回值：		成功 = 0 失败 = MBEDTLS_ERR_ECP_XXX

表 1-3 生成 ECDSA 密钥参数 API

1.2.3 哈希值(Hash value)

哈希是计算机科学中一种对数据的处理方法，通过某种特定的函数/算法(称为哈希函数/算法)将要检索的项与用来检索的索引(称为哈希，或者哈希值)关联起来，生成一种便于搜寻的数据结构(称为哈希表)。它也常用作一种信息安全的实作方法，由一串数据中经过哈希算法计算出来的数据指纹，经常用来识别档案与数据是否有被窜改，以保证档案与数据确实是由原创者所提供。哈希算法也被用来加密存在数据库中的密码字符串，由于哈希算法所计算出来的哈希值(Hash Value)具有不可逆(无法逆向演算回原本的数值)的性质，且哈希值必须随明文改变而改变，因此可有效的保护密码。例如 SHA-256 是最广泛使用的密码哈希函数之一，它生成 256 位值的哈希值[5]。

1.2.4 数字签名

数字签名的写入与生成，是将哈希函数处理后的讯息再以私钥加密，并仅能以发送方释出的公钥做解密。可于 mbedtls 函数库调用 ***mbedtls_ecdsa_write_signature()***做计算并写入签章，其中会以 ***mbedtls_ecdsa_write_signature_restartable()***逐步记录已执行程序，遇到错误将立即返回错误代码，且可于指定位置执行重启动作，以节省总阻塞时间(Total Blocking Time, TBT)。详细函数说明如表 1-4~表 1-5 所示。

mbedtls_ecdsa_write_signature(mbedtls_ecdsa_context *ctx,mbedtls_md_type_t md_alg, const unsigned char *hash, size_t hlen, unsigned char *sig, size_t sig_size, size_t *slen, int (*f_rng)(void *, unsigned char *, size_t), void *p_rng)		
参数：	*ctx	ECDSA 密钥参数
	md_alg	密码哈希函数
	*hash	原始哈希值存放地址
	hlen	哈希函数长度(Bytes)
	*sig	写入签章存位
	sig_size	签章存位长度(Bytes)
	*slen	存储签名的实际长度的地址
	*f_rng	随机数函数
	*p_rng	f_rng 附加函数 (不使用则填 NULL)

[5] https://zh.wikipedia.org/zh-tw/%E6%95%A3%E5%88%97%E5%87%BD%E6%95%B8

返回值：	成功 = 0 失败 = MBEDTLS_ERR_ECP_XXX, MBEDTLS_ERR_MPI_XXX, MBEDTLS_ERR_ASN1_XXX

表 1-4 签章写入 API

mbedtls_ecdsa_write_signature_restartable(mbedtls_ecdsa_context *ctx, mbedtls_md_type_t md_alg, const unsigned char *hash, size_t hlen, unsigned char *sig, size_t sig_size, size_t *slen, int (*f_rng)(void *, unsigned char *, size_t), void *p_rng, mbedtls_ecdsa_restart_ctx *rs_ctx);		
参数：	*ctx	ECDSA 密钥参数
	md_alg	密码哈希函数
	*hash	原始哈希值存放地址
	hlen	哈希函数长度(Bytes)
	*sig	写入签章存位
	sig_size	签章存位长度(Bytes)
	*slen	存储签名的实际长度的地址
	*f_rng	随机数函数
	*p_rng	f_rng 附加函数 (不使用则填 NULL)
	*rs_ctx	重启用的密钥参数(不使用则填 NULL)
返回值：		成功 = 0 超过最大操作次数 = MBEDTLS_ERR_ECP_IN_PROGRESS 失败 = MBEDTLS_ERR_ECP_XXX, MBEDTLS_ERR_MPI_XXX, MBEDTLS_ERR_ASN1_XXX

表 1-5 签章重启 API

1.2.5 数字签名验证

由于数字签名以发送方的私钥处理资料，且仅能以发送方提供的公钥通过验证，故其常用于防止伪造、抵赖与检测窜改。如图 1-4 所示，数字签名的认证重点为：公钥解密后，所得摘要(Digest)是否与数据发送方将讯息经哈希处理后所得的摘要一致，若两者一致，代表签章合法，反之为非法。可于 mbedtls 函数库叫用 ***mbedtls_ecdsa_read_signature()***对数字签名执行验证，函数中也包含 ***mbedtls_ecdsa_write_signature_restartable()***进行辅助，详细函数说明如表 1-6~表 1-7 所示。

mbedtls_ecdsa_read_signature(mbedtls_ecdsa_context *ctx, const unsigned char *hash, size_t hlen, const unsigned char *sig, size_t slen)		
参数：	*ctx	ECDSA密钥参数
	*hash	原始哈希值存放地址
	hlen	哈希函数长度(Bytes)
	*sig	写入签章存位
	slen	存储签名的实际长度的地址
返回值：		成功 = 0 失败 = MBEDTLS_ERR_ECP_XXX, MBEDTLS_ERR_MPI_XXX

表 1-6 签章验证 API

mbedtls_ecdsa_read_signature_restartable(mbedtls_ecdsa_context *ctx, const unsigned char *hash, size_t hlen, const unsigned char *sig, size_t slen, mbedtls_ecdsa_restart_ctx *rs_ctx)		
参数：	*ctx	ECDSA密钥参数
	*hash	原始哈希值存放地址
	hlen	哈希函数长度(Bytes)
	*sig	写入签章存位
	slen	存储签名的实际长度的地址
	*rs_ctx	重启用的密钥参数(不使用则填NULL)
返回值：		成功 = 0 失败 = MBEDTLS_ERR_ECP_XXX, MBEDTLS_ERR_MPI_XXX

表 1-7 重启签章验证 API

1.3 执行结果

执行后会打印出以下信息：

```
+---------------------------------+
|       ECDSA Sample Code         |
+---------------------------------+

 ECDSA test start...

 mbedtls ECDSA init done

 mbedtls ECDSA generate key       : passed     takes 0.934 seconds,  934 ticks
 mbedtls ECDSA key pair           : passed     takes 0.3 seconds,  3 ticks
 mbedtls ECDSA signature generate : passed     takes 0.954 seconds,  954 ticks
 mbedtls ECDSA signature verify   : passed     takes 3.3791 seconds,  3791 ticks

 mbedtls ECDSA done

 ECDSA test done ...
Test Done!
```

图 1-5 执行结果

2. 代码介绍

使用 mbedtls 函数库产生 ECDSA 的公钥及私钥，代码位于 *ecdsa_test.c*。

```
    mbedtls_ecdsa_context ecdsa;
    size_t sig_len;
    const mbedtls_ecp_curve_info *curve_info =
                            mbedtls_ecp_curve_info_from_grp_id(MBEDTLS_ECP_DP_BP256R1);
    mbedtls_pk_context pk;
    mbedtls_ecp_keypair *ecp;

    mbedtls_pk_init(&pk);

    /* Initializes an ECDSA context. */
    mbedtls_ecdsa_init(&ecdsa);
    printf(" mbedtls ECDSA init done  \n\n");

    printf(" mbedtls ECDSA generate key       : ");
    /* Generate the keypair on an elliptic curve */
    ret = mbedtls_ecdsa_genkey(&ecdsa, curve_info->grp_id, myrand, NULL);
    if(ret == 0) {
        printf("passed");
        u32Time = get_timer0_counter();
        printf("     takes %d.%d seconds,  %d ticks\n", u32Time / 1000000, u32Time / 1000,
g_u32Ticks);
    } else {
        printf("failed! ret[%d]\n", ret);
        mbedtls_ecdsa_free(&ecdsa);
        return ret;
    }

printf(" mbedtls ECDSA key pair           : ");
/* Generate the public key and private key from key pair */
    ret = mbedtls_ecdsa_from_keypair(&ecdsa, &ecdsa);
    if(ret == 0) {
        printf("passed");
        u32Time = get_timer0_counter();
        printf("     takes %d.%d seconds,  %d ticks\n", u32Time / 1000000, u32Time / 1000,
g_u32Ticks);
    } else {
        printf("failed! ret[%d]\n", ret);
        mbedtls_ecdsa_free(&ecdsa);
        return ret;
}
```

密钥完成后，选用所需哈希函数将数据加密，最后再以私钥制作数字签名。代码位于 ecdsa_test.c。

```
/* Cacualate the data hash */
    const mbedtls_md_info_t *md_info;
    md_info = mbedtls_md_info_from_type(MBEDTLS_MD_SHA256);
    mbedtls_md(md_info,(const unsigned char *) test_image_bin, sizeof(test_image_bin),
hash);
```

```
    /* Computes the ECDSA signature and writes it to a buffer */
    printf(" mbedtls ECDSA signature generate : ");
    ret = mbedtls_ecdsa_write_signature(&ecdsa, MBEDTLS_MD_SHA256, hash, sizeof(hash),
tmp, sizeof(tmp), &sig_len, myrand, NULL);

    if(ret == 0) {
        printf("passed");
        u32Time = get_timer0_counter();
        printf("     takes %d.%d seconds,  %d ticks\n", u32Time / 1000000, u32Time / 1000,
g_u32Ticks);
    } else {
        printf("failed! ret[%d]\n", ret);
        mbedtls_ecdsa_free(&ecdsa);
        return ret;
    }
```

读取数字签名内容，并使用公钥执行认证。代码位于 *ecdsa_test.c*。

```
  /* Reads and verifies an ECDSA signature */
    printf(" mbedtls ECDSA signature verify   : ");
    ret = mbedtls_ecdsa_read_signature(&ecdsa, hash, sizeof(hash), tmp, sig_len);
    if(ret == 0) {
        printf("passed");
        u32Time = get_timer0_counter();
        printf("     takes %d.%d seconds,  %d ticks\n", u32Time / 1000000, u32Time / 1000,
g_u32Ticks);
    } else {
        printf("failed! ret[%d]\n", ret);
        mbedtls_ecdsa_free(&ecdsa);
        return ret;
    }

    mbedtls_ecdsa_free(&ecdsa);
    mbedtls_ecdsa_free(ecp);
    printf("\n mbedtls ECDSA done  \n");
```

3. 软件与硬件需求

3.1 软件需求

- BSP 版本
  - M4521_BSP_CMSIS_V3.00.003
- IDE 版本
  - Keil uVersion 6.10

3.2 硬件需求

- 电路组件
  - NuTiny-SDK-M4521S

4. 目录信息

EC_ M4521_ECC_Certification _V1.00	
Library	Sample code header and source files
CMSIS	Cortex® Microcontroller Software Interface Standard (CMSIS) by Arm® Corp.
Device	CMSIS compliant device header file
StdDriver	All peripheral driver header and source files
SampleCode	
ExampleCode	Source file of example code
ThirdParty	
mbedtls-3.1.0	A portable, easy to use, readable and flexible SSL library. Mbed TLS is available as open source under the Apache 2.0 license or the GPL 2.0 license. Its official website: https://tls.mbed.org/

图 4-1 目录信息

5. 范例程序执行

1. 根据目录信息章节进入 ExampleCode 路径中的 KEIL 文件夹，双击 *ECC_Certification.uvproj*。
2. 进入编译模式界面
   - 编译
   - 下载代码至内存
   - 进入 / 离开仿真模式
3. 进入仿真模式界面
   - 执行代码

6. 修订纪录

Date	Revision	Description
2022.08.01	1.00	初始发布。

Important Notice

Nuvoton Products are neither intended nor warranted for usage in systems or equipment, any malfunction or failure of which may cause loss of human life, bodily injury or severe property damage. Such applications are deemed, "Insecure Usage".

Insecure usage includes, but is not limited to: equipment for surgical implementation, atomic energy control instruments, airplane or spaceship instruments, the control or operation of dynamic, brake or safety systems designed for vehicular use, traffic signal instruments, all types of safety devices, and other applications intended to support or sustain life.

All Insecure Usage shall be made at customer's risk, and in the event that third parties lay claims to Nuvoton as a result of customer's Insecure Usage, customer shall indemnify the damages and liabilities thus incurred by Nuvoton.

Please note that all data and specifications are subject to change without notice.
All the trademarks of products and companies mentioned in this datasheet belong to their respective owners.