M4521 RSA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	本范例于 M4521 开发板上透过 mbedtls 函数库，以纯软件实现 RSA 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 RSA 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 RSA 加密算法(Rivest–Shamir–Adleman Encryption Algorithm)

RSA 是由极大整数无法进行因子分解的数学难题而衍生，对极大整数做因子分解的难度决定了 RSA 的可靠性，其愈困难，算法则愈可靠。RSA 是种非对称加密算法，电子商业中被广泛使用，它需要两个密钥:公开密钥(Public key)与私有密钥(private key)；公钥用作加密，私钥则用作解密。使用公钥把明文加密后所得的密文，只能用相对应的私钥才能解密并得到原本的明文，最初用来加密的公钥不能用作解密。反之，也能使用私钥把明文加密，再用公钥解密。

由于加解密需要两个不同的密钥，故被称为非对称加密。公钥可以公开，可任意向外发布；私钥不可以公开，必须由使用者自行严格秘密保管，绝不透过任何途径向任何人提供，也不会透露给被信任的要通讯的另一方。且基于公开密钥加密的特性，它还能提供数字签名的功能，使数字档案可以得到如同在纸本档案上亲笔签署的效果[1]。

[1] https://zh.wikipedia.org/zh-tw/%E5%85%AC%E5%BC%80%E5%AF%86%E9%92%A5%E5%8A%A0%E5%AF%86

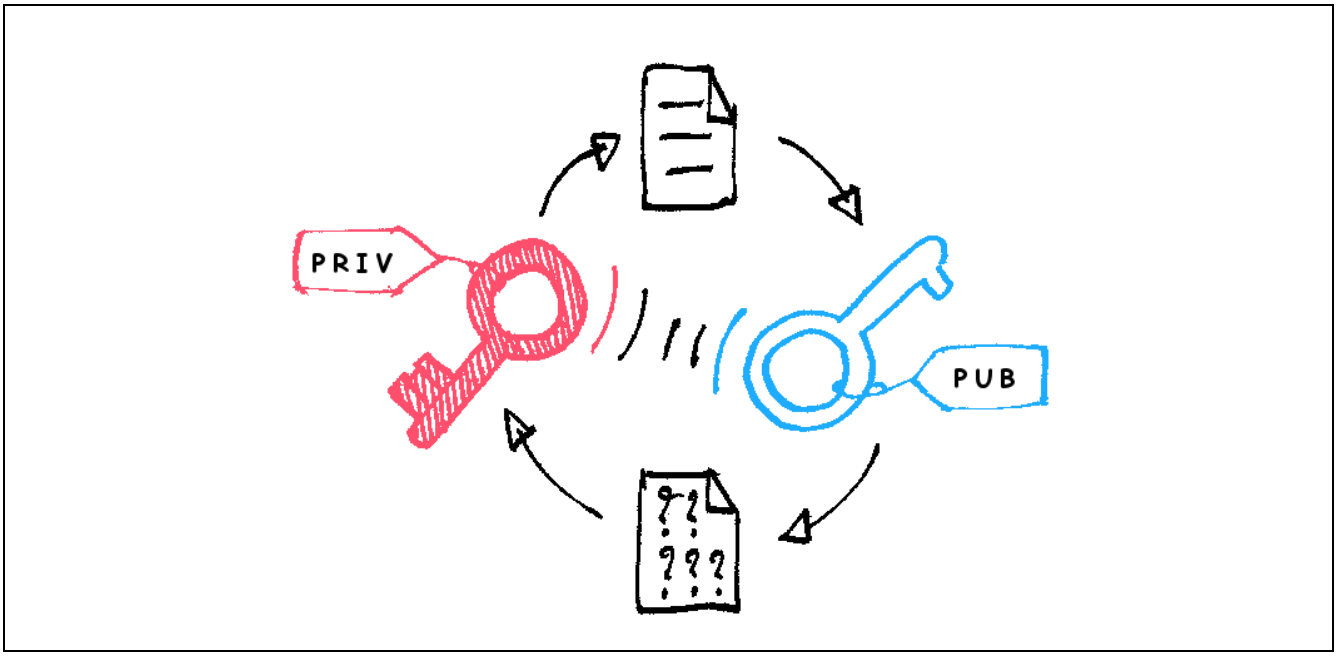

图 1- 1 单向加解密流程示意[2]

1.1.1 数字签名(Digital Signature)

数字签名是一种类似于写在纸上的普通签名、但是使用了公钥加密领域的技术，用于鉴别数字讯息的方法。一套数字签名通常会定义两种互补的运算，一个用于签名，另一个用于验证。法律用语中的数字签名与数字签名代表之意义并不相同。电子签章指的是依附于数字文件并与其相关连，用以辨识及确认数字文件签署人身份、资格及数字文件真伪者；数字签名则是以数学算法或其他方式运算对其加密而形成的签章[3]。

通常会使用公钥加密，用私钥解密。而在数字签名中，会使用私钥加密(相当于生成签名)，公钥解密(相当于验证签名)。可以直接对讯息进行签名(即使用私钥加密，此时加密的目的是为了签名，而不是保密)，验证者用公钥正确解密讯息，如果和原讯息一致，则验证签名成功。但通常会对讯息的哈希值签名，因为通常哈希值的长度远小于讯息原文，使得签名(非对称加密)的效率大大提高。

[2] https://zh.wikipedia.org/zh-tw/%E5%85%AC%E5%BC%80%E5%AF%86%E9%92%A5%E5%8A%A0%E5%AF%86
[3] https://zh.wikipedia.org/zh-tw/%E6%95%B8%E4%BD%8D%E7%B0%BD%E7%AB%A0

1.2 RSA 原理与操作

1.2.1 RSA 密钥产生

若想要通过一个不可靠的媒体接收讯息，可以用以下的方式来产生公钥与私钥：

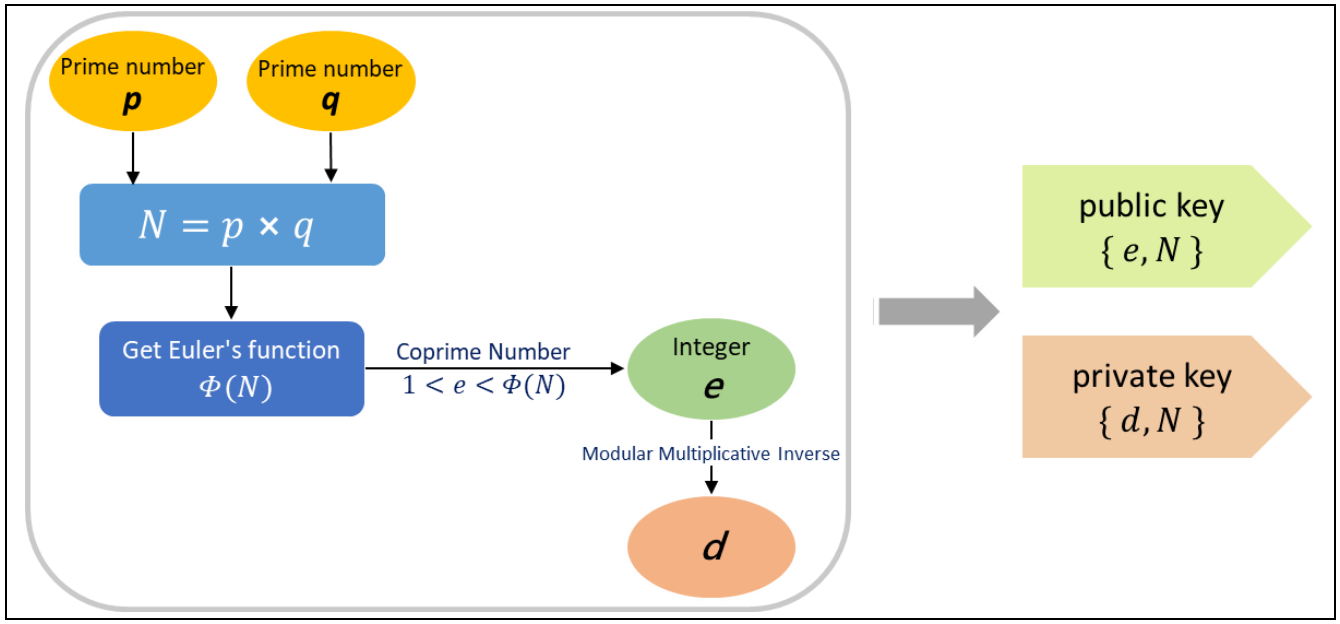

图 1- 2 RSA 密钥产生流程

1. 任一选出两个大的质数p与q，且$p \neq q$，并计算乘积$N = pq$
2. 根据欧拉函数，求得$r = \varphi(N) = \varphi(p)\varphi(q) = (p-1)(q-1)$
3. 选择一个小于r的整数e，使e与r互质，并满足条件$1 < e < r$。接着求得e对于r的模反元素，命名为d (需满足$ed \equiv 1 (\mathrm{mod}\ r)$)
4. 将p与q的记录销毁

其中(N, e)是公钥、(N, d)是私钥，使用者可将其公钥传送给要接收加密讯息的对象，并将私钥藏起。

如果任何人使用公钥加密明文，所得密文可透过不安全途径(例如网络)传送，只有持有对应私钥的使用者才能解密得到明文；其他人即使从网络上窃取到密文及加密公钥，也几乎无法强行解得明文。

1.2.2 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[4]。

1.2.3 加密

PEM (Privacy-Enhanced Mail)是储存、传输密码学的密钥、公钥凭证和其他数据的文件格式的业界标准[5]，因此公钥与私钥最常见扩展名也为.pem。针对 PEM 格式的密钥，可于 mbedtls 函数库调用 **mbedtls_rsa_pkcs1_encrypt()**对数据进行加密运算，其定义如表 1-1：

mbedtls_rsa_pkcs1_encrypt(mbedtls_rsa_context *ctx, int (*f_rng)(void *, unsigned char *, size_t), void *p_rng, size_t ilen, const unsigned char *input, unsigned char *output)		
参数：	*ctx	RSA 密钥参数
	*f_rng	随机数函数
	*p_rng	f_rng 附加函数(不使用则填 NULL)
	ilen	明文长度(Bytes)
	*input	待加密明文
	*output	密文存放地址
返回值：		成功 = 0 失败 = MBEDTLS_ERR_RSA_XXX

表 1-1 API_mbedtls_rsa_pkcs1_encrypt()

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

[5] https://zh.wikipedia.org/zh-tw/Pem%E6%A0%BC%E5%BC%8F

1.2.4 解密

得到密文讯息后，便可利用持有私钥，于 mbedtls 函数库内调用 ***mbedtls_rsa_pkcs1_decrypt()***进行解密运算，其定义如表 1-2：

mbedtls_rsa_pkcs1_decrypt(mbedtls_rsa_context *ctx, int (*f_rng)(void *, unsigned char *, size_t), void *p_rng, size_t *olen, const unsigned char *input, unsigned char *output, size_t output_max_len)		
参数：	*ctx	RSA 密钥参数
	*f_rng	随机数函数
	*p_rng	f_rng 附加函数 (不使用则填 NULL)
	*olen	储存明文长度的地址
	*input	待解密密文
	*output	明文存放地址
返回值：		成功 = 0 失败 = MBEDTLS_ERR_RSA_XXX

表 1-2 API_mbedtls_rsa_pkcs1_decrypt()

1.2.5 数字签名认证

RSA 也可用于讯息署名，当客户想传递署名讯息给用户时，可为其讯息计算一个哈希值，接着用私钥「加密」(步骤如同 1.2.3 节所示)此哈希值，并将此「署名」加在讯息的后面。

署名讯息只能使用客户的公钥解密，当用户收到讯息后，便可利用公钥「解密」(步骤如 1.2.4 节所示)哈希值，然后将这笔数据与用户自己为此讯息计算的哈希值相比较。假如两者相符的话，即可得知发信人持有客户的私钥，且此讯息在传播过程中没被篡改过。

mbedtls 函数库内可调用 ***mbedtls_rsa_pkcs1_sign()***，以 RSA 私钥进行签章，其定义如表 1-3。

mbedtls_rsa_pkcs1_sign(mbedtls_rsa_context *ctx, int (*f_rng)(void *, unsigned char *, size_t), void *p_rng, mbedtls_md_type_t md_alg, unsigned int hashlen, const unsigned char *hash, unsigned char *sig)		
参数：	*ctx	RSA 密钥参数
	*f_rng	随机数函数
	*p_rng	f_rng 附加函数 (不使用则填 NULL)
	md_alg	密码哈希函数
	hashlen	哈希函数长度(Bytes)

	*hash	原始哈希值存放地址
	*sig	签章存放地址，必须为长度 ctx->len 字节的可读地址
返回值：		成功 = 0 失败 = MBEDTLS_ERR_RSA_XXX

表 1‑3 API_mbedtls_rsa_pkcs1_sign()

mbedtls 函数库可调用 ***mbedtls_rsa_pkcs1_verify()***进行 RSA 公钥操作与认证，定义如表 1‑4。

mbedtls_rsa_pkcs1_verify(mbedtls_rsa_context *ctx, mbedtls_md_type_t md_alg, unsigned int hashlen, const unsigned char *hash, const unsigned char *sig)		
参数：	*ctx	RSA 密钥参数
	md_alg	密码哈希函数
	hashlen	哈希函数长度(Bytes)
	*hash	原始哈希值存放地址
	*sig	签章存放地址，必须为长度 ctx->len 字节的可读地址
返回值：		成功 = 0 失败 = MBEDTLS_ERR_RSA_XXX

表 1‑4 API_mbedtls_rsa_pkcs1_verify()

1.3 执行结果

执行后会打印出以下信息：

```
MBEDTLS RSA self test ...
Pure software crypto running.

  RSA encrypt with hash test start

  mbedtls_pk_parse_key      : passed     takes 0.30 seconds,  30 ticks
  mbedtls_rsa_check_privkey : passed     takes 0.63 seconds,  63 ticks
  PKCS#1 encryption       : passed     takes 0.197 seconds,  197 ticks
  PKCS#1 decryption       : passed     takes 7.7783 seconds,  7783 ticks
  PKCS#1 data sign        : passed     takes 6.6554 seconds,  6554 ticks
  PKCS#1 sig. verify      : passed     takes 0.156 seconds,  156 ticks


  RSA encrypt with hash test done

  Test OK
```

图 1- 3 执行结果

2. 代码介绍

使用 mbedtls 函数库执行密钥确认，代码位于 *rsa_test.c*。

```
int ret = 0;
/* Read the RSA key */
if((ret = mbedtls_pk_parse_key(
                  &pk,
                  (const unsigned char *)SSL_USER_PRIV_KEY_PEM,
                  sizeof(SSL_USER_PRIV_KEY_PEM),
                  NULL,  0,
                 myrand, NULL)) != 0)
   {
        printf("  failed\n ! mbedtls_pk_parse_key returned %d\n", ret);
        return -1;
   }
/* check RSA privkey key */
mbedtls_rsa_context *rsa = mbedtls_pk_rsa(pk);
if(mbedtls_rsa_check_privkey(rsa) != 0)
{
     printf("  failed\n !mbedtls_rsa_check_privkey\n");
     return -1;
}
/* check RSA pubkey key */
if(mbedtls_rsa_check_pubkey(rsa) != 0)
{
     printf("  failed\n !mbedtls_rsa_check_pubkey\n");
     return -1;
}
printf(" passed");
g_u32Time = get_timer0_counter();

/* TIMER0->CNT is the elapsed us */
printf("     takes %d.%d seconds,  %d ticks\n", g_u32Time / 1000000, g_u32Time / 1000,
g_u32Ticks);
}
```

密钥读取成功后，开始执行加解密测试。代码位于 *rsa_test.c*。

```
printf("  PKCS#1 encryption      : ");
 /* RSA encryption */
/* This function adds the message padding, then performs an RSA operation */
if(mbedtls_rsa_pkcs1_encrypt(rsa, myrand, NULL,
                                 PT_LEN, rsa_plaintext,
                                 rsa_ciphertext) != 0)
    {
        if(verbose != 0)
            printf("failed\n");

        ret = 1;
        goto cleanup;
    }

    if(verbose != 0)
```

```
    {
        printf("passed");
        g_u32Time = get_timer0_counter();
        printf("     takes %d.%d seconds,  %d ticks\n", g_u32Time / 1000000, g_u32Time /
1000, g_u32Ticks);
    }

    enable_sys_tick(1000);
    start_timer0();
    printf("  PKCS#1 decryption      : ");
    /* RSA decryption */
/* This function performs an RSA operation, then removes the message padding */
    if(mbedtls_rsa_pkcs1_decrypt(rsa, myrand, NULL,
                                 &len, rsa_ciphertext, rsa_decrypted,
                                 sizeof(rsa_decrypted)) != 0)
    {
        if(verbose != 0)
            printf("failed\n");

        ret = 1;
        goto cleanup;
    }

    if(memcmp(rsa_decrypted, rsa_plaintext, len) != 0)
    {
        if(verbose != 0)
            printf("failed\n");

        ret = 1;
        goto cleanup;
    }

    if(verbose != 0)
    {
        printf("passed");
        g_u32Time = get_timer0_counter();
        printf("     takes %d.%d seconds,  %d ticks\n", g_u32Time / 1000000, g_u32Time /
1000, g_u32Ticks);
    }
```

以 RSA 私钥进行签章，并使用公钥执行签章认证。代码位于 *rsa_test.c*。

```
if(verbose != 0)
        printf("  PKCS#1 data sign       : ");

    if(mbedtls_sha1(test_image_bin, sizeof(test_image_bin), sha1sum) != 0)
{
        if(verbose != 0)
            printf("failed\n");

        return(1);
    }
    /* performs a private RSA operation to sign a message digest using PKCS#1 */
    if(mbedtls_rsa_pkcs1_sign(rsa, myrand, NULL,
                              MBEDTLS_MD_SHA1, sha_set_value,
                              sha1sum, rsa_ciphertext) != 0)
```

```
    {
        if(verbose != 0)
            printf("failed\n");

        ret = 1;
        goto cleanup;
    }

    if(verbose != 0)
    {
        printf("passed");
        g_u32Time = get_timer0_counter();

        printf("     takes %d.%d seconds,  %d ticks\n", g_u32Time / 1000000, g_u32Time /
1000, g_u32Ticks);
    }
    enable_sys_tick(1000);
    start_timer0();

    printf("  PKCS#1 sig. verify     : ");
    /* performs a public RSA operation and checks the message digest */
    if(mbedtls_rsa_pkcs1_verify(rsa, MBEDTLS_MD_SHA1, sha_set_value,
                                sha1sum, rsa_ciphertext) != 0)
    {
        if(verbose != 0)
            printf("failed\n");

        ret = 1;
        goto cleanup;
    }

    if(verbose != 0)
    {
        printf("passed");
        g_u32Time = get_timer0_counter();
        printf("     takes %d.%d seconds,  %d ticks\n", g_u32Time / 1000000, g_u32Time /
1000, g_u32Ticks);
    }
```

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_RSA_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 *RSA_Certification.uvproj*。
2. 进入编译模式接口
   - 编译
   - 下载代码至内存
   - 进入 / 离开除错模式
3. 进入除错模式接口
   - 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.