M4521 RSA認證

NuMicro® 32位系列微控制器範例代碼介紹

文件資訊

應用簡述	本範例於 M4521 開發板上透過 mbedtls 函數庫，以純軟體實現 RSA 加解密運算功能
BSP 版本	M4521_BSP_CMSIS_V3.00.003
開發平台	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 許可下以開源形式提供的 SSL(Secure Sockets Layer)函數庫，含有豐富的加解密函式可用來進行安全通訊，避免資料遭竊聽。

此範例程式利用 mbedtls 函數庫，以純軟體執行 RSA 金鑰加解密運算，以緩解部分開發板由於設計規格不同，而無法以硬體實現加解密功能的問題。

1.1 RSA 加密演算法(Rivest–Shamir–Adleman Encryption Algorithm)

RSA 是由極大整數無法進行因數分解的數學難題而衍生，對極大整數做因數分解的難度決定了 RSA 的可靠性，其愈困難，演算法則愈可靠。RSA 是種非對稱加密演算法，電子商業中被廣泛使用，它需要兩個金鑰:公開密鑰(Public key)與私有密鑰(private key)；公鑰用作加密，私鑰則用作解密。使用公鑰把明文加密後所得的密文，只能用相對應的私鑰才能解密並得到原本的明文，最初用來加密的公鑰不能用作解密。反之，也能使用私鑰把明文加密，再用公鑰解密。

由於加解密需要兩個不同的密鑰，故被稱為非對稱加密。公鑰可以公開，可任意向外發佈；私鑰不可以公開，必須由使用者自行嚴格秘密保管，絕不透過任何途徑向任何人提供，也不會透露給被信任的要通訊的另一方。且基於公開密鑰加密的特性，它還能提供數位簽章的功能，使數位檔案可以得到如同在紙本檔案上親筆簽署的效果[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

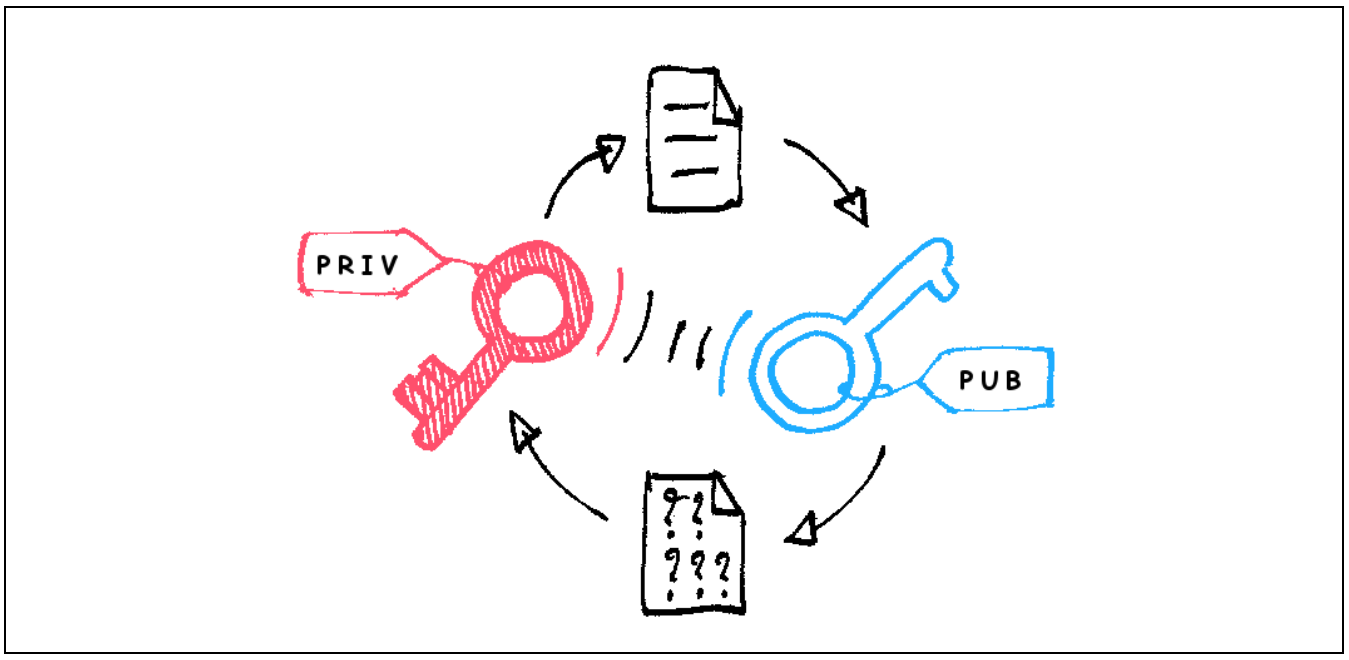

圖 1-1 單向加解密流程示意[2]

1.1.1 數位簽章(Digital Signature)

數位簽章是一種類似於寫在紙上的普通簽名、但是使用了公鑰加密領域的技術，用於鑑別數位訊息的方法。一套數位簽章通常會定義兩種互補的運算，一個用於簽名，另一個用於驗證。法律用語中的數位簽章與數位簽章代表之意義並不相同。電子簽章指的是依附於數位文件並與其相關連，用以辨識及確認數位文件簽署人身分、資格及數位文件真偽者；數位簽章則是以數學演算法或其他方式運算對其加密而形成的簽章[3]。

通常會使用公鑰加密，用私鑰解密。而在數位簽章中，會使用私鑰加密(相當於生成簽名)，公鑰解密(相當於驗證簽名)。可以直接對訊息進行簽名(即使用私鑰加密，此時加密的目的是為了簽名，而不是保密)，驗證者用公鑰正確解密訊息，如果和原訊息一致，則驗證簽名成功。但通常會對訊息的雜湊值簽名，因為通常雜湊值的長度遠小於訊息原文，使得簽名(非對稱加密)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 原理與操作

1.2.1 RSA 金鑰產生

若想要通過一個不可靠的媒體接收訊息，可以用以下的方式來產生公鑰與私鑰：

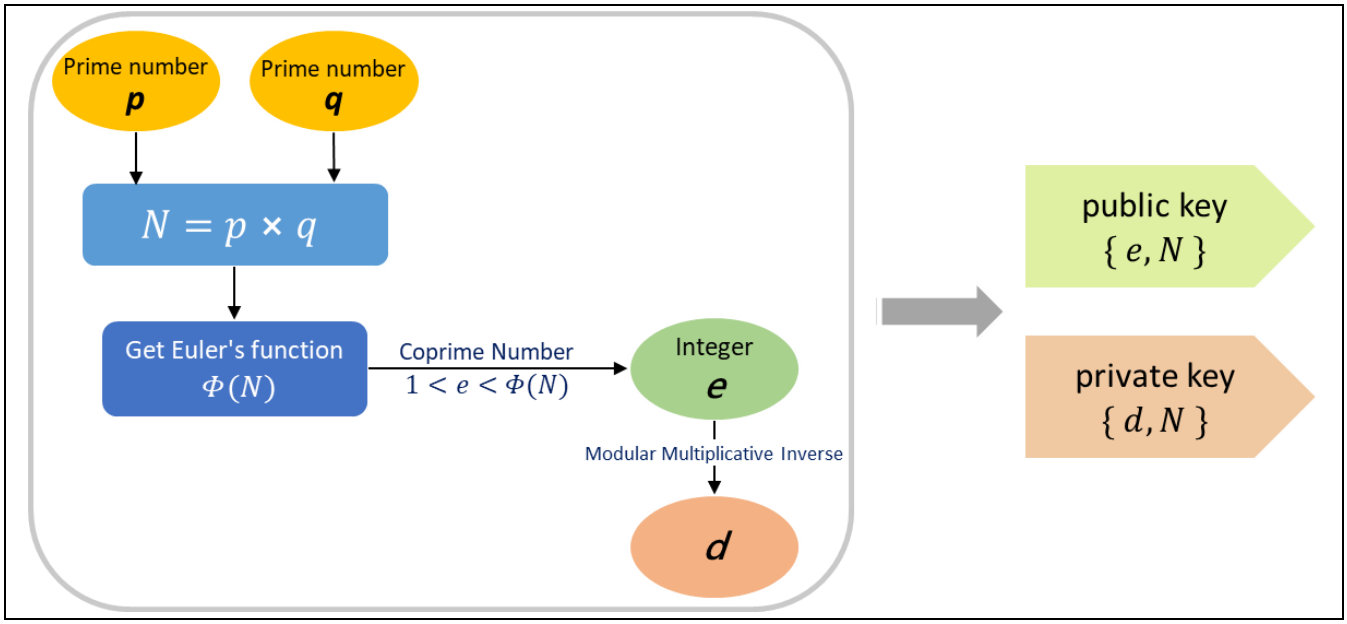

圖 1-2 RSA 金鑰產生流程

1. 任一選出兩個大的質數p與q，且$p \neq q$，並計算乘積$N = pq$
2. 根據歐拉函數，求得$r = \varphi(N) = \varphi(p)\varphi(q) = (p-1)(q-1)$
3. 選擇一個小於r的整數e，使e與r互質，並滿足條件$1 < e < r$。接著求得e對於r的模反元素，命名為d (需滿足$ed \equiv 1 (\mathrm{mod}\ r)$)
4. 將p與q的記錄銷毀

其中(N, e)是公鑰、(N, d)是私鑰，使用者可將其公鑰傳送給要接收加密訊息的對象，並將私鑰藏起。

如果任何人使用公鑰加密明文，所得密文可透過不安全途徑(例如網路)傳送，只有持有對應私鑰的使用者才能解密得到明文；其他人即使從網路上竊取到密文及加密公鑰，也幾乎無法強行解得明文。

1.2.2 雜湊值(Hash value)

雜湊是電腦科學中一種對資料的處理方法，通過某種特定的函式/演算法(稱為雜湊函式/演算法)將要檢索的項與用來檢索的索引(稱為雜湊，或者雜湊值)關聯起來，生成一種便於搜尋的資料結構(稱為雜湊表)。它也常用作一種資訊安全的實作方法，由一串資料中經過雜湊演算法計算出來的資料指紋，經常用來識別檔案與資料是否有被竄改，以保證檔案與資料確實是由原創者所提供。雜湊演算法也被用來加密存在資料庫中的密碼字串，由於雜湊演算法所計算出來的雜湊值(Hash Value)具有不可逆(無法逆向演算回原本的數值)的性質，且雜湊值必須隨明文改變而改變，因此可有效的保護密碼。例如 SHA-256 是最廣泛使用的密碼雜湊函式之一，它生成 256 位元值的雜湊值[4]。

1.2.3 加密

PEM (Privacy-Enhanced Mail)是儲存、傳輸密碼學的金鑰、公開金鑰憑證和其他資料的檔案格式的業界標準[5]，因此公鑰與私鑰最常見副檔名也為.pem。針對 PEM 格式的金鑰，可於 mbedtls 函數庫呼叫 **mbedtls_rsa_pkcs1_encrypt()**對資料進行加密運算，其定義如表 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)		
參數：	*ctx	RSA 金鑰參數
	*f_rng	亂數函數
	*p_rng	f_rng 附加函數(不使用則填 NULL)
	ilen	明文長度(Bytes)
	*input	待加密明文
	*output	密文存放位址
回傳值：		成功 = 0 失敗 = 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

得到密文訊息後，便可利用持有私鑰，於 mbedtls 函數庫內呼叫 ***mbedtls_rsa_pkcs1_decrypt ()***進行解密運算，其定義如表 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)		
參數：	*ctx	RSA 金鑰參數
	*f_rng	亂數函數
	*p_rng	f_rng 附加函數 (不使用則填 NULL)
	*olen	儲存明文長度的位址
	*input	待解密密文
	*output	明文存放位址
回傳值：		成功 = 0 失敗 = MBEDTLS_ERR_RSA_XXX

表 1-2 API_mbedtls_rsa_pkcs1_decrypt()

1.2.5 數位簽章認證

RSA 也可用於訊息署名，當客戶想傳遞署名訊息給使用者時，可為其訊息計算一個雜湊值，接著用私鑰「加密」(步驟如同 1.2.3 節所示)此雜湊值，並將此「署名」加在訊息的後面。

署名訊息只能使用客戶的公鑰解密，當使用者收到訊息後，便可利用公鑰「解密」(步驟如 1.2.4 節所示)雜湊值，然後將這筆資料與使用者自己為此訊息計算的雜湊值相比較。假如兩者相符的話，即可得知發信人持有客戶的私鑰，且此訊息在傳播過程中沒被篡改過。

mbedtls 函數庫內可呼叫 ***mbedtls_rsa_pkcs1_sign()***，以 RSA 私鑰進行簽章，其定義如表 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)		
參數：	*ctx	RSA 金鑰參數

	*f_rng	亂數函數
	*p_rng	f_rng 附加函數 (不使用則填 NULL)
	md_alg	密碼雜湊函數
	hashlen	雜湊函數長度(Bytes)
	*hash	原始雜湊值存放位址
	*sig	簽章存放位址，必須為長度 ctx->len 字節的可讀位址
回傳值：		成功 = 0 失敗 = MBEDTLS_ERR_RSA_XXX

表 1-3 API_mbedtls_rsa_pkcs1_sign()

mbedtls 函數庫可呼叫 **mbedtls_rsa_pkcs1_verify()**進行 RSA 公鑰操作與認證，定義如表 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)		
參數：	*ctx	RSA 金鑰參數
	md_alg	密碼雜湊函數
	hashlen	雜湊函數長度(Bytes)
	*hash	原始雜湊值存放位址
	*sig	簽章存放位址，必須為長度 ctx->len 字節的可讀位址
回傳值：		成功 = 0 失敗 = MBEDTLS_ERR_RSA_XXX

表 1-4 API_mbedtls_rsa_pkcs1_verify()

1.3 執行結果

執行後會打印出以下資訊：

```
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
```

圖 1-3 執行結果

2. 代碼介紹

使用 mbedtls 函數庫執行金鑰確認，代碼位於 *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);
}
```

金鑰讀取成功後，開始執行加解密測試。代碼位於 *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 私鑰進行簽章，並使用公鑰執行簽章認證。代碼位於 *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. 軟體與硬體需求

3.1 軟體需求

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

3.2 硬體需求

- 電路元件
  - NuTiny-SDK-M4521S

4. 目錄資訊

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/

圖 4-1 目錄資訊

5. 範例程式執行

1. 根據目錄資訊章節進入 ExampleCode 路徑中的 KEIL 資料夾，雙擊 *RSA_Certification.uvproj*。
2. 進入編譯模式介面
   - 編譯
   - 下載代碼至記憶體
   - 進入 / 離開除錯模式
3. 進入除錯模式介面
   - 執行代碼

6. 修訂紀錄

Date	Revision	Description
2022.07.22	1.00	初始發布。

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.