[UEFI Practice] RSA Algorithm

原创 jiangwei0512 Posted on 2019-07-17 23:39:53 Read 6.8k Collection 4 Likes 2

Category Column: UEFI Development Basics Article Tags: openssl uefi rsa

UEFI Development ... This column includes this content

136 articles Subscribe to our column

摘要 This article introduces the basic principles of the RSA algorithm and its application in Windows. The RSA algorithm is an asymmetric encryption algorithm that uses a pair of keys for encryption and decryption operations and is suitable for secure data transmission scenarios.

The summary is generated in C Know , supported by DeepSeek-R1 full version, go to experience>

Brief description of RSA algorithm

The RSA in the RSA algorithm is not an abbreviation of any professional term. It is the first letters of three names. These three people are Rivest, Shamir and Adleman, who proposed this algorithm.

The RSA algorithm is an encryption algorithm, and its use is of course to encrypt and decrypt data.

RSA is an asymmetric encryption algorithm. If there is an asymmetric algorithm, there must be a symmetric algorithm.

A symmetric algorithm means that **the key used for encryption and decryption is the same** ; while an asymmetric algorithm has a pair of keys, called **a public key and a private key** . Data encrypted by the public key is decrypted with the private key, and data encrypted by the private key is decrypted with the public key.

The advantage of an asymmetric algorithm is that both parties can hold their own public and private keys. I can give you the data encrypted with my private key directly, and you can decrypt it with my public key. The public key itself can be made public, so there is no risk. However, if it is a symmetric algorithm, I can give you the encrypted data directly after encrypting it with a secret key, but how do I give you the secret key? Because the secret key itself is private, if there is a security problem during the transmission process, the encryption itself is meaningless.

Of course, asymmetric algorithms also have disadvantages. For the same security level, asymmetric algorithm encryption time is longer than symmetric algorithm encryption time.

Finally, let's talk about public keys and private keys. They are actually a set of binary data, and the length can be 1024 or 2048 bits. It should be noted that they appear in pairs and there is a corresponding relationship between the two.

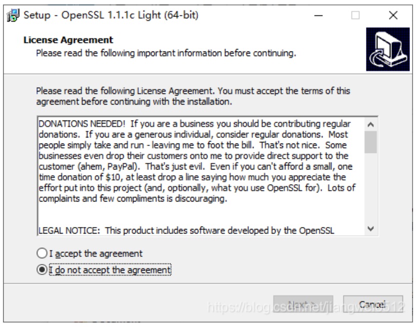
Using RSA algorithm under Windows

The RSA algorithm is included in the open source encryption library OpenSSL.

The relevant OpenSSL source code can be found at https://www.openssl.org/ .

For use under Windows, the binary installation files can be found at Win32/Win64 OpenSSL Installer for Windows - Shining Light Productions .

Install as follows:



Setup - OpenSSL 1.1.1c Light (64-bit)

License Agreement

Please read the following important information before continuing.

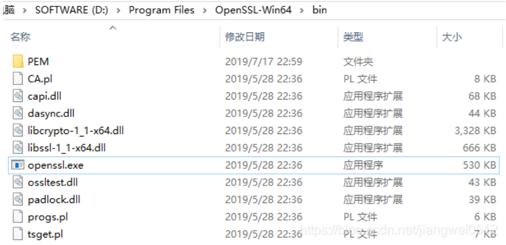
Please read the following License Agreement. You must accept the terms of this agreement before continuing with the installation.

DONATIONS NEEDED! If you are a business you should be contributing regular donations. If you are a generous individual, consider regular donations. Most people simply take and run - leaving me to foot the bill. That's not nice. Some businesses even drop their customers onto me to provide direct support to the customer (ahem, PayPal). That's just evil. Even if you can't afford a small, one time donation of $10, at least drop a line saying how much you appreciate the effort put into this project (and, optionally, what you use OpenSSL for). Lots of complaints and few compliments is discouraging.

LEGAL NOTICE: This product includes software developed by the OpenSSL

○ I accept the agreement
● I do not accept the agreement

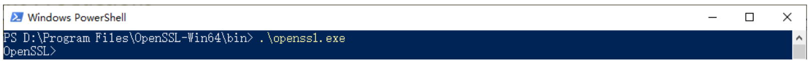
After the installation is complete, you can find the corresponding files in the directory:



脑 › SOFTWARE (D:) › Program Files › OpenSSL-Win64 › bin

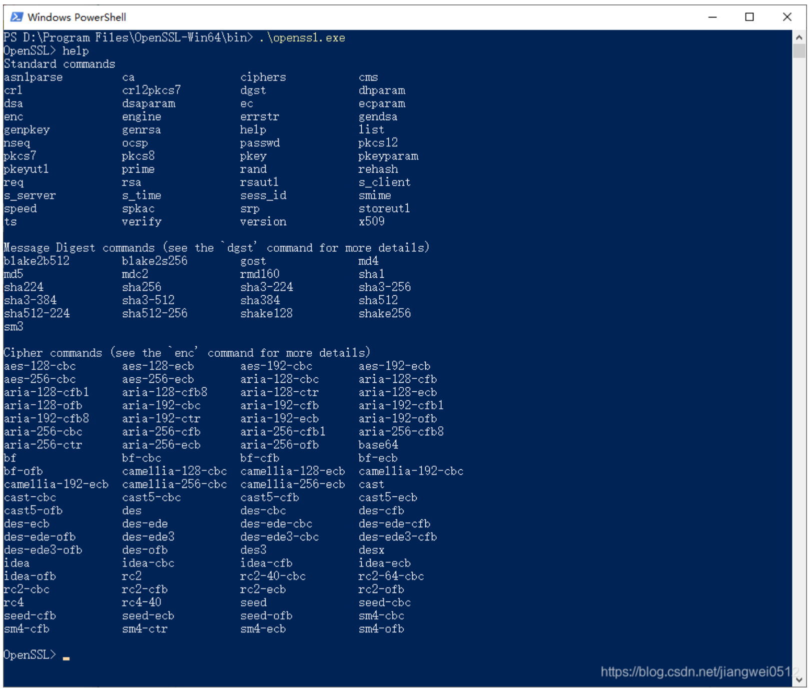
名称	修改日期	类型	大小
PEM	2019/7/17 22:59	文件夹	
CA.pl	2019/5/28 22:36	PL 文件	8 KB
capi.dll	2019/5/28 22:36	应用程序扩展	68 KB
dasync.dll	2019/5/28 22:36	应用程序扩展	44 KB
libcrypto-1_1-x64.dll	2019/5/28 22:36	应用程序扩展	3,328 KB
libssl-1_1-x64.dll	2019/5/28 22:36	应用程序扩展	666 KB
openssl.exe	2019/5/28 22:36	应用程序	530 KB
ossltest.dll	2019/5/28 22:36	应用程序扩展	43 KB
padlock.dll	2019/5/28 22:36	应用程序扩展	39 KB
progs.pl	2019/5/28 22:36	PL 文件	6 KB
tsget.pl	2019/5/28 22:36	PL 文件	7 KB

Open a shell in the above directory and run openssl.exe:



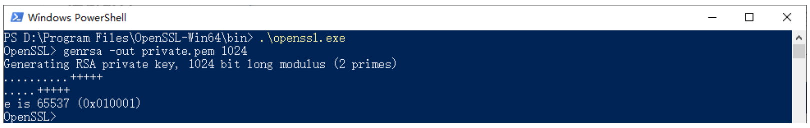
```
PS D:\Program Files\OpenSSL-Win64\bin> .\openssl.exe
OpenSSL>
```

You can view the help through help:



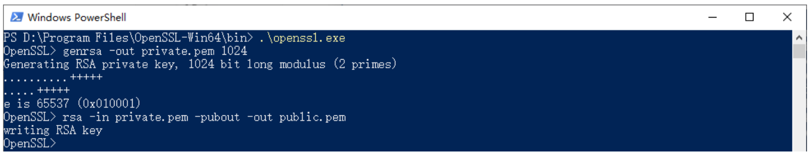
```
PS D:\Program Files\OpenSSL-Win64\bin> .\openssl.exe
OpenSSL> help
Standard commands
asn1parse         ca                ciphers           cms
crl               crl2pkcs7         dgst              dhparam
dsa               dsaparam          ec                ecparam
enc               engine            errstr            gendsa
genpkey           genrsa            help              list
nseq              ocsp              passwd            pkcs12
pkcs7             pkcs8             pkey              pkeyparam
pkeyutl           prime             rand              rehash
req               rsa               rsautl            s_client
s_server          s_time            sess_id           smime
speed             spkac             srp               storeutl
ts                verify            version           x509

Message Digest commands (see the `dgst' command for more details)
blake2b512        blake2s256        gost              md4
md5               mdc2              rmd160            sha1
sha224            sha256            sha3-224          sha3-256
sha3-384          sha3-512          sha384            sha512
sha512-224        sha512-256        shake128          shake256
sm3

Cipher commands (see the `enc' command for more details)
aes-128-cbc       aes-128-ecb       aes-192-cbc       aes-192-ecb
aes-256-cbc       aes-256-ecb       aria-128-cbc      aria-128-cfb
aria-128-cfb1     aria-128-cfb8     aria-128-ctr      aria-128-ecb
aria-128-ofb      aria-192-cbc      aria-192-cfb      aria-192-cfb1
aria-192-cfb8     aria-192-ctr      aria-192-ecb      aria-192-ofb
aria-256-cbc      aria-256-cfb      aria-256-cfb1     aria-256-cfb8
aria-256-ctr      aria-256-ecb      aria-256-ofb      base64
bf                bf-cbc            bf-cfb            bf-ecb
bf-ofb            camellia-128-cbc  camellia-128-ecb  camellia-192-cbc
camellia-192-ecb  camellia-256-cbc  camellia-256-ecb  cast
cast-cbc          cast5-cbc         cast5-cfb         cast5-ecb
cast5-ofb         des               des-cbc           des-cfb
des-ecb           des-ede           des-ede-cbc       des-ede-cfb
des-ede-ofb       des-ede3          des-ede3-cbc      des-ede3-cfb
des-ede3-ofb      des-ofb           des3              desx
idea              idea-cbc          idea-cfb          idea-ecb
idea-ofb          rc2               rc2-40-cbc        rc2-64-cbc
rc2-cbc           rc2-cfb           rc2-ecb           rc2-ofb
rc4               rc4-40            seed              seed-cbc
seed-cfb          seed-ecb          seed-ofb          sm4-cbc
sm4-cfb           sm4-ctr           sm4-ecb           sm4-ofb

OpenSSL> _
```

The following command can generate a private key (named private.pem):



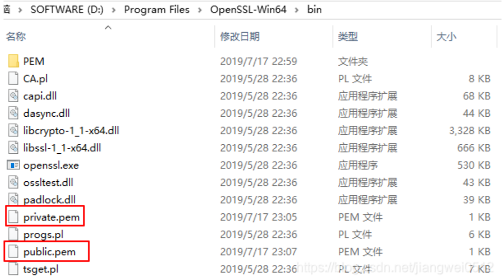
```
PS D:\Program Files\OpenSSL-Win64\bin> .\openssl.exe
OpenSSL> genrsa -out private.pem 1024
Generating RSA private key, 1024 bit long modulus (2 primes)
.........+++++
....+++++
e is 65537 (0x010001)
OpenSSL>
```

The public key can be extracted from the private key:



```
PS D:\Program Files\OpenSSL-Win64\bin> .\openssl.exe
OpenSSL> genrsa -out private.pem 1024
Generating RSA private key, 1024 bit long modulus (2 primes)
.........+++++
....+++++
e is 65537 (0x010001)
OpenSSL> rsa -in private.pem -pubout -out public.pem
writing RSA key
OpenSSL>
```

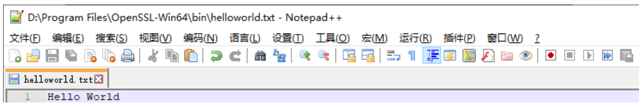
These two files can be found in the same directory:



名称	修改日期	类型	大小
PEM	2019/7/17 22:59	文件夹	
CA.pl	2019/5/28 22:36	PL 文件	8 KB
capi.dll	2019/5/28 22:36	应用程序扩展	68 KB
dasync.dll	2019/5/28 22:36	应用程序扩展	44 KB
libcrypto-1_1-x64.dll	2019/5/28 22:36	应用程序扩展	3,328 KB
libssl-1_1-x64.dll	2019/5/28 22:36	应用程序扩展	666 KB
openssl.exe	2019/5/28 22:36	应用程序	530 KB
ossltest.dll	2019/5/28 22:36	应用程序扩展	43 KB
padlock.dll	2019/5/28 22:36	应用程序扩展	39 KB
private.pem	2019/7/17 23:05	PEM 文件	1 KB
progs.pl	2019/5/28 22:36	PL 文件	6 KB
public.pem	2019/7/17 23:07	PEM 文件	1 KB
tsget.pl	2019/5/28 22:36	PL 文件	7 KB

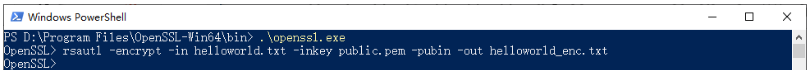
With this pair of keys, files can be encrypted and decrypted.

For example, here is a file helloworld.txt, which contains the sentence "Hello World":


D:\Program Files\OpenSSL-Win64\bin\helloworld.txt - Notepad++
文件(F) 编辑(E) 搜索(S) 视图(V) 编码(N) 语言(L) 设置(T) 工具(O) 宏(M) 运行(R) 插件(P) 窗口(W) ?
helloworld.txt

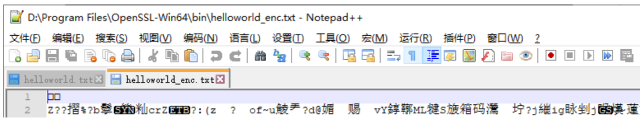
```
Hello World
```

Use the following command to encrypt with the public key:



```
PS D:\Program Files\OpenSSL-Win64\bin> .\openssl.exe
OpenSSL> rsautl -encrypt -in helloworld.txt -inkey public.pem -pubin -out helloworld_enc.txt
OpenSSL>
```

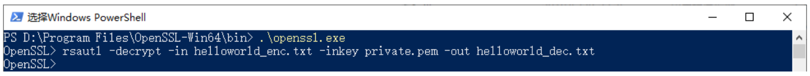
View the encrypted file:


D:\Program Files\OpenSSL-Win64\bin\helloworld_enc.txt - Notepad++
文件(F) 编辑(E) 搜索(S) 视图(V) 编码(N) 语言(L) 设置(T) 工具(O) 宏(M) 运行(R) 插件(P) 窗口(W) ?
helloworld.txt
helloworld_enc.txt

```
□□
Z??摺%?b鑾SYN籼crZETB?:(z  ?  of~u鮁弄?d@媚  赐  vY鎛鞹ML犍S旇箱码濡  圬?j繼ig脉剉jGS僕蓮
```

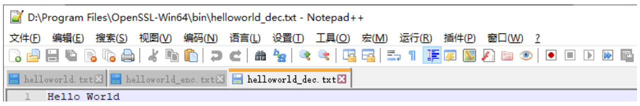
Anyway, I can't understand what it is.

Then decrypt it using the private key:



```
PS D:\Program Files\OpenSSL-Win64\bin> .\openssl.exe
OpenSSL> rsautl -decrypt -in helloworld_enc.txt -inkey private.pem -out helloworld_dec.txt
OpenSSL>
```

After decryption, we can get the original "Hello World" again:


D:\Program Files\OpenSSL-Win64\bin\helloworld_dec.txt - Notepad++
文件(F) 编辑(E) 搜索(S) 视图(V) 编码(N) 语言(L) 设置(T) 工具(O) 宏(M) 运行(R) 插件(P) 窗口(W) ?
helloworld.txt
helloworld_enc.txt
helloworld_dec.txt

```
Hello World
```

Finally, let's check the public and private keys just generated. In fact, they can both be opened using text tools.

This is the private key:

bash AI generated projects 登录复制

```
-----BEGIN RSA PRIVATE KEY-----
MIICXAIBAAKBgQDYjsDqqWseZhgnksNmFpFpsii5mBHT2p99CDe5jONLk3hOflSJ
LiEVvRfxaW4bP/ahUm/4YedBOAWx/qIbp/y33Lry3hCKlcw0S7CDLOuyJYTyR9P4
DV6OdsJbHtd/KMt1oI0MIp5TUeNMx28LpT0mkAV9xRC30w5z9rsPpO96BwIDAQAB
AoGAayWmmfSdRXsqNFpPR7Ge3PcDY2C0YWfoRYrNV4c35ureOagT15P1VrYqpI2P
pvfs5UDOmcyHCSPY3YqpVpm1VbJJ8wpiWboQtQhEZ6Z0faENiFjzYxgY77H5BLh6
9X/MH97ug4ByMkhVQFS5cfAwbyVsw54fj42zKkT5G5e8tXECQQDzAenHtjEQ8KsM
3DwPy98zKLO1l/HhhL9wYVF7HjAeBtrFHnw1LaTz9MKh6Ow/qh78LhEtXYcKVQE9
D3VfsJNtAkEA5CLRD/6UGIk/Lv8Zsg11tikTDPyIcPJdDAay4OlvrHFVBr2B/kpO
YGYQK6r8kT74T1SM1A4YUNOo0e2ppKEmwwJBAI6C1rk6jbfjjEy0c7zH0RPNkOa0
PzQEh8i+KezMHWfemTn00N7W79/p8KLHWJVVjWpTEdvK98EFbP6ELE973FkCQFMx
ayG1CZaE/jiKKHmotONPyTW0JaFikJHhI3wnRGUTExmZI/1yZXB756u099OTrgNn
5s8xsKZQ+UBMtc9mQPcCQGVKSD5Lov1b75el4FhsDJGh/+zhXxiMoS5pwPslC7SJ
rkV7Owkb7IM0L7ruL6j4u/tC8Rk0mJ2x4crjWOt6bl0=
-----END RSA PRIVATE KEY-----
```

收起 ︿

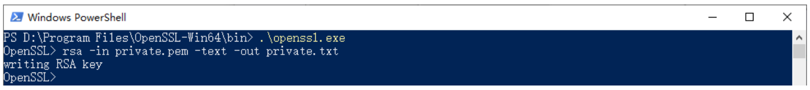
This is the public key:

bash AI generated projects 登录复制

```
-----BEGIN PUBLIC KEY-----
MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDYjsDqqWseZhgnksNmFpFpsii5
mBHT2p99CDe5jONLk3hOflSJLiEVvRfxaW4bP/ahUm/4YedBOAWx/qIbp/y33Lry
3hCKlcw0S7CDLOuyJYTyR9P4DV6OdsJbHtd/KMt1oI0MIp5TUeNMx28LpT0mkAV9
xRC30w5z9rsPpO96BwIDAQAB
-----END PUBLIC KEY-----
```

Of course, the above doesn’t show anything specific, and it doesn’t seem to be 1024 bits (it will be converted later)…

In fact, it can still be converted through commands:



```
PS D:\Program Files\OpenSSL-Win64\bin> .\openssl.exe
OpenSSL> rsa -in private.pem -text -out private.txt
writing RSA key
OpenSSL>
```

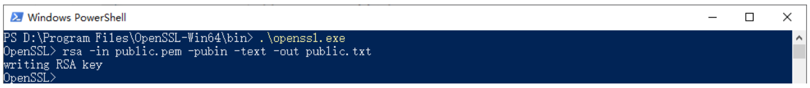
The content after conversion is as follows:

```
RSA Private-Key: (1024 bit, 2 primes)
modulus:
    00:d8:8e:c0:ea:a9:6b:1e:66:18:27:92:c3:66:16:
    91:69:b2:28:b9:98:11:d3:da:9f:7d:08:37:b9:8c:
    e3:4b:93:78:4e:7e:54:89:2e:21:15:bd:17:f1:69:
    6e:1b:3f:f6:a1:52:6f:f8:61:e7:41:38:05:b1:fe:
    a2:1b:a7:fc:b7:dc:ba:f2:de:10:8a:95:cc:34:4b:
    b0:83:2c:eb:b2:25:84:f2:47:d3:f8:0d:5e:8e:76:
    c2:5b:1e:d7:7f:28:cb:75:a0:8d:0c:22:9e:53:51:
    e3:4c:c7:6f:0b:a5:3d:26:90:05:7d:c5:10:b7:d3:
    0e:73:f6:bb:0f:a4:ef:7a:07
publicExponent: 65537 (0x10001)
privateExponent:
    6b:25:a6:99:f4:9d:45:7b:2a:34:5a:4f:47:b1:9e:
    dc:f7:03:63:60:b4:61:67:e8:45:8a:cd:57:87:37:
    e6:ea:de:39:a8:13:d7:93:f5:56:b6:2a:a4:8d:8f:
    a6:f7:ec:e5:40:ce:99:cc:87:09:23:d8:dd:8a:a9:
    56:99:b5:55:b2:49:f3:0a:62:59:ba:10:b5:08:44:
    67:a6:74:7d:a1:0d:88:58:f3:63:18:18:ef:b1:f9:
    04:b8:7a:f5:7f:cc:1f:de:ee:83:80:72:32:48:55:
    40:54:b9:71:f0:30:6f:25:6c:c3:9e:1f:8f:8d:b3:
    2a:44:f9:1b:97:bc:b5:71
prime1:
    00:f3:01:e9:c7:b6:31:10:f0:ab:0c:dc:3c:0f:cb:
    df:33:28:b3:b5:97:f1:e1:84:bf:70:61:51:7b:1e:
    30:1e:06:da:c5:1e:7c:35:2d:a4:f3:f4:c2:a1:e8:
    ec:3f:aa:1e:fc:2e:11:2d:5d:87:0a:55:01:3d:0f:
    75:5f:b0:93:6d
prime2:
    00:e4:22:d1:0f:fe:94:18:89:3f:2e:ff:19:b2:0d:
    75:b6:29:13:0c:fc:88:70:f2:5d:0c:06:b2:e0:e9:
    6f:ac:71:55:06:bd:81:fe:4a:4e:60:66:10:2b:aa:
    fc:91:3e:f8:4f:54:8c:d4:0e:18:50:d3:a8:d1:ed:
    a9:a4:a1:26:c3
exponent1:
    00:8e:82:d6:b9:3a:8d:b7:e3:8c:4c:b4:73:bc:c7:
    d1:13:cd:90:e6:b4:3f:34:04:87:c8:be:29:ec:cc:
    1d:67:de:99:39:f4:d0:de:d6:ef:df:e9:f0:a2:c7:
    58:95:55:8d:6a:53:11:db:ca:f7:c1:05:6c:fe:84:
    2c:4f:7b:dc:59
exponent2:
    53:31:6b:21:b5:09:96:84:fe:38:8a:28:79:a8:b4:
    e3:4f:c9:35:b4:25:a1:62:90:91:e1:23:7c:27:44:
    65:13:13:19:99:23:fd:72:65:70:7b:e7:ab:b4:f7:
    d3:93:ae:03:67:e6:cf:31:b0:a6:50:f9:40:4c:b5:
    cf:66:40:f7
coefficient:
    65:4a:48:3e:4b:a2:fd:5b:ef:97:a5:e0:58:6c:0c:
    91:a1:ff:ec:e1:5f:18:8c:a1:2e:69:c0:fb:25:0b:
    b4:89:ae:45:7b:3b:09:1b:ec:83:34:2f:ba:ee:2f:
    a8:f8:bb:fb:42:f1:19:34:98:9d:b1:e1:ca:e3:58:
    eb:7a:6e:5d
-----BEGIN RSA PRIVATE KEY-----
MIICXAIBAAKBgQDYjsDqqWseZhgnksNmFpFpsii5mBHT2p99CDe5jONLk3hOflSJ
LiEVvRfxaW4bP/ahUm/4YedBOAWx/qIbp/y33Lry3hCKlcw0S7CDLOuyJYTyR9P4
DV6OdsJbHtd/KMt1oI0MIp5TUeNMx28LpT0mkAV9xRC30w5z9rsPpO96BwIDAQAB
AoGAayWmmfSdRXsqNFpPR7Ge3PcDY2C0YWfoRYrNV4c35ureOagT15P1VrYqpI2P
pvfs5UDOmcyHCSPY3YqpVpm1VbJJ8wpiWboQtQhEZ6Z0faENiFjzYxgY77H5BLh6
9X/MH97ug4ByMkhVQFS5cfAwbyVsw54fj42zKkT5G5e8tXECQQDzAenHtjEQ8KsM
3DwPy98zKLO1l/HhhL9wYVF7HjAeBtrFHnw1LaTz9MKh6Ow/qh78LhEtXYcKVQE9
```

```
D3VfsJNtAkEA5CLRD/6UGIk/Lv8Zsg11tikTDPyIcPJdDAay4OlvrHFVBr2B/kpO
YGYQK6r8kT74T1SM1A4YUNOo0e2ppKEmwwJBAI6C1rk6jbfjjEy0c7zH0RPNkOa0
PzQEh8i+KezMHWfemTn00N7W79/p8KLHWJVVjWpTEdvK98EFbP6ELE973FkCQFMx
ayG1CZaE/jiKKHmotONPyTW0JaFikJHhI3wnRGUTExmZI/1yZXB756u099OTrgNn
5s8xsKZQ+UBMtc9mQPcCQGVKSD5Lov1b75el4FhsDJGh/+zhXxiMoS5pwPslC7SJ
rkV7Owkb7IM0L7ruL6j4u/tC8Rk0mJ2x4crjWOt6bl0=
-----END RSA PRIVATE KEY-----
```

收起 ︿

Of course the public key can also be converted into a string:



```
PS D:\Program Files\OpenSSL-Win64\bin> .\openssl.exe
OpenSSL> rsa -in public.pem -pubin -text -out public.txt
writing RSA key
OpenSSL>
```

The obtained content is as follows:

bash AI generated projects 登录复制

```bash
RSA Public-Key: (1024 bit)
Modulus:
    00:d8:8e:c0:ea:a9:6b:1e:66:18:27:92:c3:66:16:
    91:69:b2:28:b9:98:11:d3:da:9f:7d:08:37:b9:8c:
    e3:4b:93:78:4e:7e:54:89:2e:21:15:bd:17:f1:69:
    6e:1b:3f:f6:a1:52:6f:f8:61:e7:41:38:05:b1:fe:
    a2:1b:a7:fc:b7:dc:ba:f2:de:10:8a:95:cc:34:4b:
    b0:83:2c:eb:b2:25:84:f2:47:d3:f8:0d:5e:8e:76:
    c2:5b:1e:d7:7f:28:cb:75:a0:8d:0c:22:9e:53:51:
    e3:4c:c7:6f:0b:a5:3d:26:90:05:7d:c5:10:b7:d3:
    0e:73:f6:bb:0f:a4:ef:7a:07
Exponent: 65537 (0x10001)
-----BEGIN PUBLIC KEY-----
MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDYjsDqqWseZhgnksNmFpFpsii5
mBHT2p99CDe5jONLk3hOflSJLiEVvRfxaW4bP/ahUm/4YedBOAWx/qIbp/y33Lry
3hCKlcw0S7CDLOuyJYTyR9P4DV6OdsJbHtd/KMt1oI0MIp5TUeNMx28LpT0mkAV9
xRC30w5z9rsPpO96BwIDAQAB
-----END PUBLIC KEY-----
```

收起 ︿

There is a bunch of data here, which is exactly 128 8-bit data, that is, 1024 bits (note that the first 00 does not need to be paid attention to).

These data will be used in the actual calculation, which mainly refers to the use in the code, when they are a number, such as in the following function:

cpp AI generated projects 登录复制 run

```cpp
/**
  Sets the tag-designated key component into the established RSA context.

  This function sets the tag-designated RSA key component into the established
  RSA context from the user-specified non-negative integer (octet string format
  represented in RSA PKCS#1).
  If BigNumber is NULL, then the specified key component in RSA context is cleared.

  If RsaContext is NULL, then return FALSE.

  @param[in, out]  RsaContext  Pointer to RSA context being set.
  @param[in]       KeyTag      Tag of RSA key component being set.
  @param[in]       BigNumber   Pointer to octet integer buffer.
                               If NULL, then the specified key component in RSA
                               context is cleared.
  @param[in]       BnSize      Size of big number buffer in bytes.
                               If BigNumber is NULL, then it is ignored.

  @retval  TRUE   RSA key component was set successfully.
  @retval  FALSE  Invalid RSA key component tag.

**/
BOOLEAN
EFIAPI
RsaSetKey (
  IN OUT  VOID         *RsaContext,
  IN      RSA_KEY_TAG  KeyTag,
  IN      CONST UINT8  *BigNumber,
  IN      UINTN        BnSize
  )
```

收起 ︿

Here, BigNumber is the array converted from the above data. Arrays have different types, represented by enumerations:

cpp AI generated projects 登录复制 run

```cpp
///
/// RSA Key Tags Definition used in RsaSetKey() function for key component identification.
///
typedef enum {
  RsaKeyN,      ///< RSA public Modulus (N)
  RsaKeyE,      ///< RSA Public exponent (e)
  RsaKeyD,      ///< RSA Private exponent (d)
  RsaKeyP,      ///< RSA secret prime factor of Modulus (p)
  RsaKeyQ,      ///< RSA secret prime factor of Modules (q)
  RsaKeyDp,     ///< p's CRT exponent (== d mod (p - 1))
  RsaKeyDq,     ///< q's CRT exponent (== d mod (q - 1))
  RsaKeyQInv    ///< The CRT coefficient (== 1/q mod p)
} RSA_KEY_TAG;
```

收起 ︿

The above comments can correspond to the converted content.

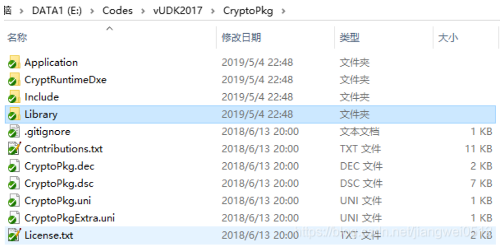
RSA algorithm under UEFI

The above content allows us to know some basic knowledge and operation methods of RSA.

The following describes how to use the RSA algorithm under UEFI.

OpenSSL under UEFI

There is a package under UEFI called CryptoPkg:

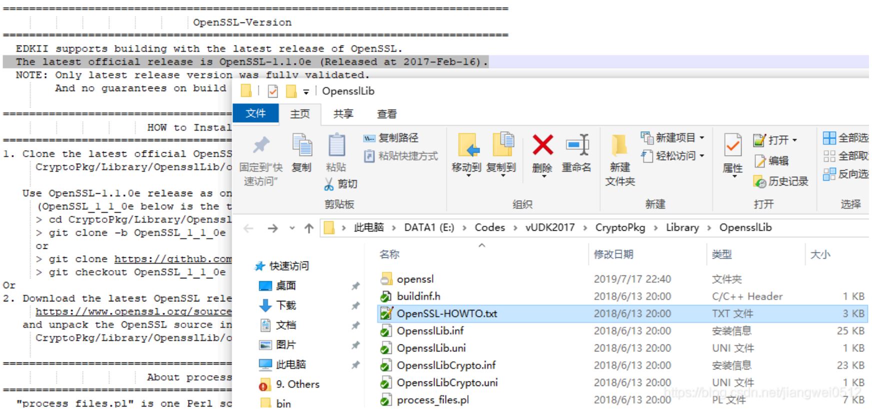


脑 › DATA1 (E:) › Codes › vUDK2017 › CryptoPkg ›

名称	修改日期	类型	大小
Application	2019/5/4 22:48	文件夹	
CryptRuntimeDxe	2019/5/4 22:48	文件夹	
Include	2019/5/4 22:48	文件夹	
Library	2019/5/4 22:48	文件夹	
.gitignore	2018/6/13 20:00	文本文档	1 KB
Contributions.txt	2018/6/13 20:00	TXT 文件	11 KB
CryptoPkg.dec	2018/6/13 20:00	DEC 文件	2 KB
CryptoPkg.dsc	2018/6/13 20:00	DSC 文件	7 KB
CryptoPkg.uni	2018/6/13 20:00	UNI 文件	1 KB
CryptoPkgExtra.uni	2018/6/13 20:00	UNI 文件	1 KB
License.txt	2018/6/13 20:00	TXT 文件	2 KB

This includes the OpenSSL library.

However, there is no default code, so you need to download it separately:



```
==============================================================================
                          OpenSSL-Version
==============================================================================
  EDKII supports building with the latest release of OpenSSL.
  The latest official release is OpenSSL-1.1.0e (Released at 2017-Feb-16).
  NOTE: Only latest release version was fully validated.
        And no guarantees on build

=====================================
                     HOW to Instal
=====================================
1. Clone the latest official OpenSS
     CryptoPkg/Library/OpensslLib/o

   Use OpenSSL-1.1.0e release as on
     (OpenSSL_1_1_0e below is the t
     > cd CryptoPkg/Library/Openssl
     > git clone -b OpenSSL_1_1_0e
     or
     > git clone https://github.com
     > git checkout OpenSSL_1_1_0e
Or
2. Download the latest OpenSSL rele
     https://www.openssl.org/source
   and unpack the OpenSSL source in
     CryptoPkg/Library/OpensslLib/o

=====================================
                     About process
=====================================
  "process files.pl" is one Perl sc
```

此电脑 › DATA1 (E:) › Codes › vUDK2017 › CryptoPkg › Library › OpensslLib

名称	修改日期	类型	大小
openssl	2019/7/17 22:40	文件夹	
buildinf.h	2018/6/13 20:00	C/C++ Header	1 KB
OpenSSL-HOWTO.txt	2018/6/13 20:00	TXT 文件	3 KB
OpensslLib.inf	2018/6/13 20:00	安装信息	25 KB
OpensslLib.uni	2018/6/13 20:00	UNI 文件	1 KB
OpensslLibCrypto.inf	2018/6/13 20:00	安装信息	23 KB
OpensslLibCrypto.uni	2018/6/13 20:00	UNI 文件	1 KB
process_files.pl	2018/6/13 20:00	PL 文件	7 KB

For details, please refer to OpenSSL-HOWTO.txt.

After downloading, put it in the directory shown above, name it openssl, and then you can compile it.

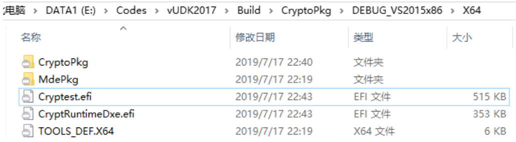
The compilation instructions are as follows:

bash AI generated projects 登录复制

```bash
build -p CryptoPkg/CryptoPkg.dsc -a X64 -t VS2015x86
```

Note that the compilation tool can be specified according to actual conditions.

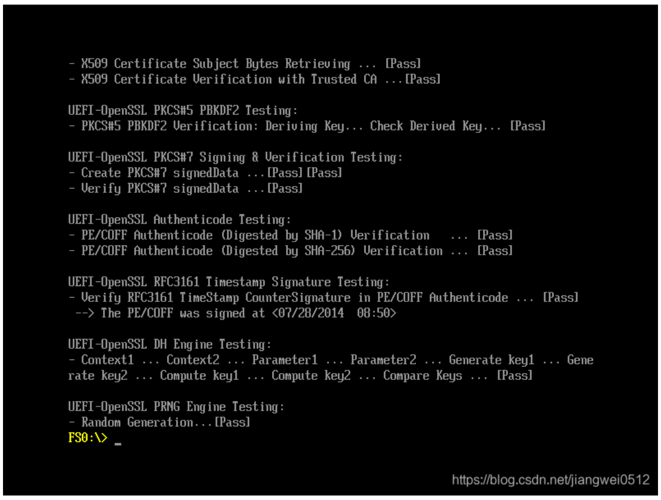
After successful compilation, you can find the generated content in the Build directory:



电脑 › DATA1 (E:) › Codes › vUDK2017 › Build › CryptoPkg › DEBUG_VS2015x86 › X64

名称	修改日期	类型	大小
CryptoPkg	2019/7/17 22:40	文件夹	
MdePkg	2019/7/17 22:19	文件夹	
Cryptest.efi	2019/7/17 22:43	EFI 文件	515 KB
CryptRuntimeDxe.efi	2019/7/17 22:43	EFI 文件	353 KB
TOOLS_DEF.X64	2019/7/17 22:19	X64 文件	6 KB

There is a Cryptest.efi that can be put into Shell for testing:



```
- X509 Certificate Subject Bytes Retrieving ... [Pass]
- X509 Certificate Verification with Trusted CA ...[Pass]

UEFI-OpenSSL PKCS#5 PBKDF2 Testing:
- PKCS#5 PBKDF2 Verification: Deriving Key... Check Derived Key... [Pass]

UEFI-OpenSSL PKCS#7 Signing & Verification Testing:
- Create PKCS#7 signedData ...[Pass][Pass]
- Verify PKCS#7 signedData ...[Pass]

UEFI-OpenSSL Authenticode Testing:
- PE/COFF Authenticode (Digested by SHA-1) Verification   ... [Pass]
- PE/COFF Authenticode (Digested by SHA-256) Verification ... [Pass]

UEFI-OpenSSL RFC3161 Timestamp Signature Testing:
- Verify RFC3161 TimeStamp CounterSignature in PE/COFF Authenticode ... [Pass]
 --> The PE/COFF was signed at <07/28/2014  08:50>

UEFI-OpenSSL DH Engine Testing:
- Context1 ... Context2 ... Parameter1 ... Parameter2 ... Generate key1 ... Gene
rate key2 ... Compute key1 ... Compute key2 ... Compare Keys ... [Pass]

UEFI-OpenSSL PRNG Engine Testing:
- Random Generation...[Pass]
FS0:\> _
```

The picture above is its test results.

As for its implementation code, it can also be found in Cryptest.inf.

Regarding the RSA algorithm, in the following code:

cpp AI generated projects 登录复制 run

```cpp
Status = ValidateCryptRsa ();
if (EFI_ERROR (Status)) {
  return Status;
}

Status = ValidateCryptRsa2 ();
if (EFI_ERROR (Status)) {
  return Status;
}
```

The above is a brief introduction to the RSA algorithm.

about Us Careers Business Cooperation Seeking coverage ☎ 400-660-0108 ✉ kefu@csdn.net Online Customer Service Working hours 8:30-22:00
Public Security Registration Number 11010502030143 Beijing ICP No. 19004658 Beijing Internet Publishing House [2020] No. 1039-165
Commercial website registration information Beijing Internet Illegal and Harmful Information Reporting Center Parental Control
Online 110 Alarm Service China Internet Reporting Center Chrome Store Download Account Management Specifications
Copyright and Disclaimer Copyright Complaints Publication License Business license