



南开大学  
Nankai University

南开大学

计算机学院和密码与网络空间安全学院

《数据安全》课程作业

---

在 OpenSSL 中进行数据签名及验证

---

姓名：陆皓喆

学号：2211044

专业：信息安全

指导教师：刘哲理

2025 年 3 月 4 日

# 目录

<b>1 实验要求</b>	<b>2</b>
<b>2 github 仓库</b>	<b>2</b>
<b>3 实验原理</b>	<b>2</b>
3.1 公钥基础设施 (PKI) . . . . .	2
3.2 OpenSSL . . . . .	2
3.2.1 对称加密 . . . . .	2
3.2.2 非对称加密 . . . . .	3
3.2.3 哈希算法 . . . . .	3
3.2.4 数字证书 . . . . .	3
3.3 数字签名及应用 . . . . .	3
<b>4 实验过程</b>	<b>3</b>
4.1 OpenSSL 安装 . . . . .	4
4.2 使用 OpenSSL 命令加解密文件 . . . . .	4
4.2.1 文件加密 . . . . .	4
4.2.2 文件解密 . . . . .	5
4.3 加解密程序编写 . . . . .	5
4.4 使用 OpenSSL 命令签名并验证 . . . . .	8
4.5 数字签名程序编写 . . . . .	9
<b>5 实验心得与体会</b>	<b>15</b>

## 1 实验要求

1. 使用 OpenSSL 命令签名并验证;
2. 运行数字签名程序, 并验证。

## 2 github 仓库

本次实验的有关代码和文件, 都已经上传至我的个人 github 中。  
您可以通过访问[此链接](#)来查阅我的代码文件。

## 3 实验原理

### 3.1 公钥基础设施 (PKI)

公钥基础设施 (PKI, Public Key Infrastructure), 是一种遵循既定标准的密钥管理平台, 它能够对所有网络应用提供加密和数字签名等密码服务及所必需的密钥和证书管理体系。简单来说, PKI 就是利用公钥理论和技术建立的提供安全服务的基础设施。

数字证书是指在互联网通讯中标志通讯各方身份信息的一个数字认证, 人们可以在网上用它来识别对方的身份。在 PKI 体系中, 建有证书管理机构 CA (Certificate Authority)。CA 中心的公钥是公开的, 因此由 CA 中心签发的内容均可以验证。

密钥的生存周期包括: 密钥的产生和登记、密钥分发、密钥更新、密钥撤销、密钥销毁等。在产生密钥后, 公钥需要在 PKI 中登记, 并通过 CA 中心的私钥签名后形成公钥证书。由于 CA 中心的公钥公开, 用户可以方便地对公钥证书进行验证, 并通过公钥证书来互相交换自己的公钥。进而, PKI 作为安全基础设施, 能够提供身份认证、数据完整性、数据保密性、数据公正性、不可抵赖性和时间戳六种安全服务。

PKI 的应用非常广泛, 为网上金融、网上银行、网上证券、电子商务、电子政务等网络中的数据交换提供了完备的安全服务功能。

OpenSSL 库提供了相关的基本功能支撑。

### 3.2 OpenSSL

OpenSSL 是一个开源的加密库, 提供了各种加密算法、数字证书管理和 SSL/TLS 协议实现等功能。其主要实验原理涉及密码学中的对称加密、非对称加密、哈希算法和数字证书等技术, 以下是具体介绍:

#### 3.2.1 对称加密

- **原理:** 对称加密使用相同的密钥进行加密和解密操作。在 OpenSSL 中, 常见的对称加密算法如 AES (高级加密标准)、DES (数据加密标准) 等都有实现。发送方使用选定的对称密钥对明文数据进行加密, 生成密文。接收方使用相同的密钥对密文进行解密, 还原出明文。
- **应用场景:** 常用于对大量数据进行快速加密, 如网络传输中的数据加密、文件加密等。例如在 SSL/TLS 握手过程中, 协商出的会话密钥就是用于对称加密后续传输的数据。

### 3.2.2 非对称加密

- **原理**：非对称加密使用一对密钥，即公钥和私钥。公钥可以公开，用于加密数据；私钥由用户自己保存，用于解密数据。例如 RSA 算法，基于大整数分解的数学难题。发送方使用接收方的公钥对数据进行加密，只有接收方使用自己的私钥才能解密。
- **应用场景**：主要用于密钥交换、数字签名等。在 SSL/TLS 握手时，客户端和服务端使用非对称加密来安全地交换对称加密所需的密钥。

### 3.2.3 哈希算法

- **原理**：哈希算法将任意长度的数据映射为固定长度的哈希值。它具有单向性，即从数据容易计算出哈希值，但从哈希值很难反推出原始数据。而且数据的任何微小变化都会导致哈希值产生很大的变化。在 OpenSSL 中，常见的哈希算法有 MD5、SHA-1、SHA-256 等。
- **应用场景**：用于数据完整性校验、数字签名中的消息摘要等。比如在数字证书中，会对证书的内容计算哈希值，并使用私钥对哈希值进行签名，以确保证书内容的完整性和真实性。

### 3.2.4 数字证书

- **原理**：数字证书是由证书颁发机构（CA）颁发的，用于证明公钥与实体（如网站、用户等）之间的绑定关系。它包含了公钥、证书持有者的身份信息、CA 的签名等内容。CA 使用自己的私钥对证书内容进行签名，用户可以使用 CA 的公钥来验证证书的合法性和完整性。
- **应用场景**：在 SSL/TLS 协议中，服务器向客户端发送自己的数字证书，客户端通过验证证书来确认服务器的身份是否可信。

在 SSL/TLS 协议中，OpenSSL 综合运用了上述多种密码学技术，以实现安全的网络通信。其基本过程一般包括：客户端与服务端进行握手，协商加密算法、交换密钥等参数，然后使用协商好的对称加密算法对数据进行加密传输，同时利用哈希算法和数字签名来保证数据的完整性和身份的真实性。

## 3.3 数字签名及应用

私钥唯一、不公开且不可伪造的特性，使得非对称密码可以应用到数字签名中。一个拥有公钥  $k_e$  和私钥  $k_d$  的用户，实现数字签名的过程如下：

1. 拥有公钥  $k_e$  的可以用其私钥  $k_d$  执行签名算法  $S$ ，以产生信息  $m$  的签名信息  $sig$ ： $sig = S_{k_d}(m)$ 。
2. 要验证一个签名信息  $sig$  是否某用户的签名时，只需要用该用户的公钥  $k_e$  执行验签方法计算出校验值  $m'$ ： $m' = V_{k_e}(c)$ ，如果  $m' = m$ ，那么验证成功，否则失败。

## 4 实验过程

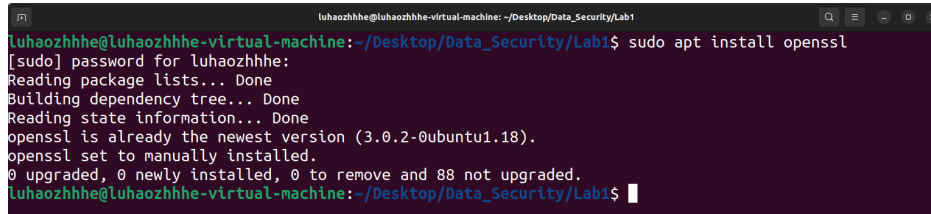
由于本次实验基于实验 2-1，所以我们先完成 2-1 的实验，然后再完成 2-2。

## 4.1 OpenSSL 安装

我们输入以下命令，来安装 OpenSSL：

```
1 sudo apt install openssl
```

我们发现，虚拟机中已经安装了 openssl。

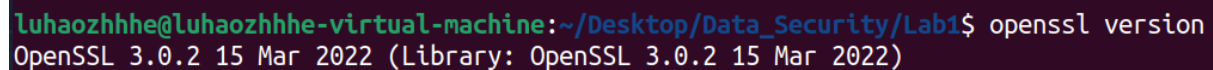


```
luhaozhhe@luhaozhhe-virtual-machine: ~/Desktop/Data_Security/Lab1
luhaozhhe@luhaozhhe-virtual-machine:~/Desktop/Data_Security/Lab1$ sudo apt install openssl
[sudo] password for luhaozhhe:
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
openssl is already the newest version (3.0.2-0ubuntu1.18).
openssl set to manually installed.
0 upgraded, 0 newly installed, 0 to remove and 88 not upgraded.
luhaozhhe@luhaozhhe-virtual-machine:~/Desktop/Data_Security/Lab1$
```

图 4.1: 安装 openssl

然后，我们输入以下命令来查看 openssl 的版本：

```
1 openssl version
```



```
luhaozhhe@luhaozhhe-virtual-machine:~/Desktop/Data_Security/Lab1$ openssl version
OpenSSL 3.0.2 15 Mar 2022 (Library: OpenSSL 3.0.2 15 Mar 2022)
```

图 4.2: openssl 版本查询

这样，我们的初始环境就搭建好了。

## 4.2 使用 OpenSSL 命令加解密文件

### 4.2.1 文件加密

首先，我们新建一个 message.txt 文件，写入以下内容：

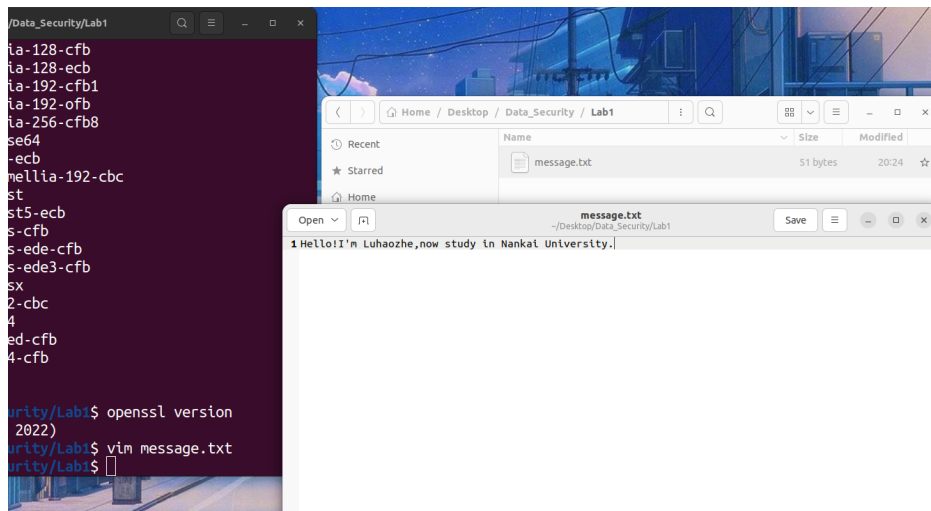


图 4.3: 输入 message.txt

接着，我们使用 aes-128-cbc 对文件进行加密，并使用 base64 编码，输出到 ciphertext.txt 中，命令行如下所示：

```
1 openssl enc -e -aes-128-cbc -in message.txt -out ciphertext.txt -K  
↪ a3171d177d1ce97ebc644ea3ff826b4e -iv 8bc65f2f883f95eea10b6f940cc805f6 -base64
```

我们发现，得到了以下的结果：

```
1 YqtYU2a9yjM8hPltn4Twr4QyA/VYVkJTFZzLvbnARjDnTH974H5+BPmhm0iQvQo1o  
2 EITamXj8Qdgv19NiV1tspQ==
```

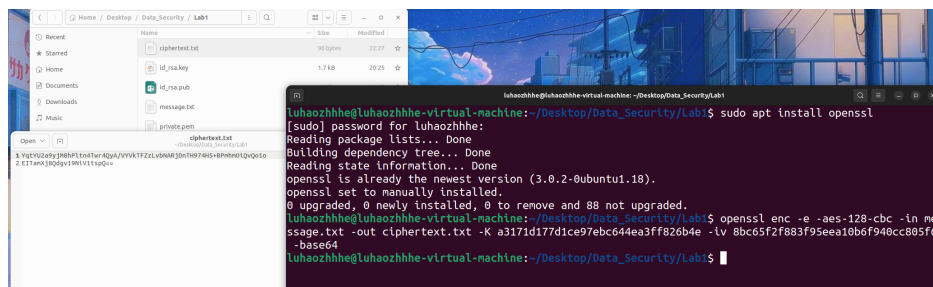


图 4.4: cbc 加密结果

#### 4.2.2 文件解密

接着，我们对加密完的文件进行解密，输入以下的命令行：

```
1 openssl enc -d -aes-128-cbc -in ciphertext.txt -out plaintext.txt -K  
↪ a3171d177d1ce97ebc644ea3ff826b4e -iv 8bc65f2f883f95eea10b6f940cc805f6 -base64
```

我们发现，得到了以下的结果：

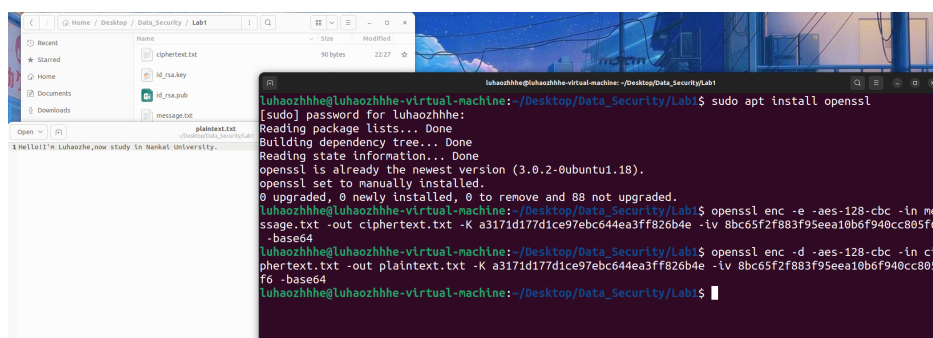


图 4.5: cbc 解密结果

我们发现，成功得到了原来的明文内容！

#### 4.3 加解密程序编写

课本上给出了 aes-128-cbc 的加解密的源代码，我们进行一下复现。

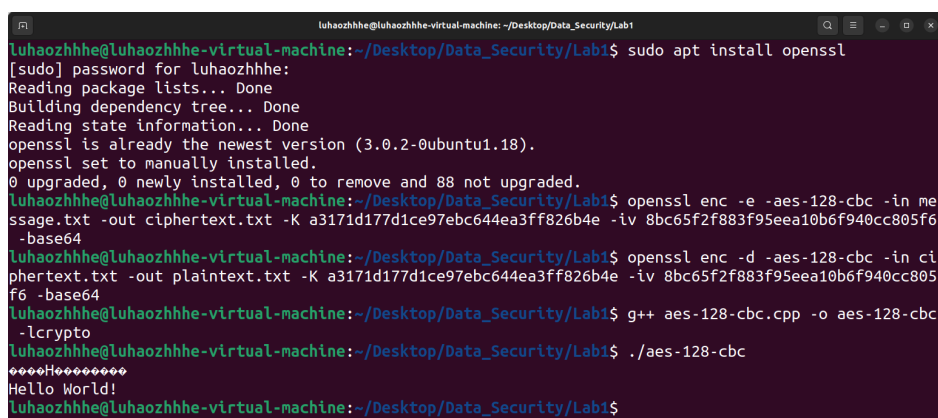
```
1  #include <stdio.h>
2  #include <string.h>
3  #include <openssl/evp.h>
4  // aes-128-cbc 加密函数
5  bool aes_128_cbc_encrypt(const uint8_t *in, int in_len, uint8_t *out, int
   ↪ *out_len, const uint8_t *key, const uint8_t *iv)
6  {
7      // 创建上下文
8      EVP_CIPHER_CTX *ctx = EVP_CIPHER_CTX_new();
9      if (!ctx)
10     {
11         return false;
12     }
13     bool ret = false;
14     // 初始化加密模块
15     if (EVP_EncryptInit_ex(ctx, EVP_aes_128_cbc(), NULL, key, iv) <= 0)
16     {
17         goto err;
18     }
19     int update_len;
20     // 向缓冲区写入数据, 同时将以对齐的数据加密并返回
21     if (EVP_EncryptUpdate(ctx, out, &update_len, in, in_len) <= 0)
22     {
23         goto err;
24     }
25     int final_len;
26     // 结束加密, 填充并返回最后的加密数据
27     if (EVP_EncryptFinal_ex(ctx, out + update_len, &final_len) <= 0)
28     {
29         goto err;
30     }
31     *out_len = update_len + final_len;
32     ret = true;
33 err:
34     EVP_CIPHER_CTX_free(ctx);
35     return ret;
36 }
37 // aes-128-cbc 解密函数, 结构与加密相似
38 bool aes_128_cbc_decrypt(const uint8_t *in, int in_len, uint8_t *out, int
   ↪ *out_len, const uint8_t *key, const uint8_t *iv)
39 {
```

```
40     EVP_CIPHER_CTX *ctx = EVP_CIPHER_CTX_new();
41     if (!ctx)
42     {
43         return false;
44     }
45     bool ret = false;
46     if (EVP_DecryptInit_ex(ctx, EVP_aes_128_cbc(), NULL, key, iv) <= 0)
47     {
48         goto err;
49     }
50     int update_len;
51     if (EVP_DecryptUpdate(ctx, out, &update_len, in, in_len) <= 0)
52     {
53         goto err;
54     }
55     int final_len;
56     if (EVP_DecryptFinal_ex(ctx, out + update_len, &final_len) <= 0)
57     {
58         goto err;
59     }
60     *out_len = update_len + final_len;
61     ret = true;
62 err:
63     EVP_CIPHER_CTX_free(ctx);
64     return ret;
65 }
66 int main()
67 {
68     // 密钥
69     uint8_t key[] = {35, 31, 71, 44, 34, 42, 76, 16, 86, 27, 93, 59, 26, 62, 4,
70                     ↪ 19};
71     // 初始化向量
72     uint8_t iv[] = {91, 66, 51, 17, 14, 40, 65, 38, 4, 60, 89, 44, 87, 63, 67, 32};
73     const char *msg = "Hello World!";
74     const int msg_len = strlen(msg);
75     // 存储密文
76     uint8_t ciphertext[32] = {0};
77     int ciphertext_len;
78     // 加密
79     aes_128_cbc_encrypt((uint8_t *)msg, msg_len, ciphertext, &ciphertext_len,
80                         ↪ (uint8_t *)key, (uint8_t *)iv);
81     printf("%s\n", ciphertext);
```



```
80 // 存储解密后的明文
81 uint8_t plaintext[32] = {0};
82 int plaintext_len;
83 // 解密
84 aes_128_cbc_decrypt((uint8_t *)ciphertext, ciphertext_len, plaintext,
85 ↪ &plaintext_len, (uint8_t *)key, (uint8_t *)iv);
86 // 输出解密后的内容
87 printf("%s\n", plaintext);
88 return 0;
}
```

我们编译 cpp 程序，运行，得到以下结果：



```
luhaozhhe@luhaozhhe-virtual-machine: ~/Desktop/Data_Security/Lab1
luhaozhhe@luhaozhhe-virtual-machine:~/Desktop/Data_Security/Lab1$ sudo apt install openssl
[sudo] password for luhaozhhe:
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
openssl is already the newest version (3.0.2-0ubuntu1.18).
openssl set to manually installed.
0 upgraded, 0 newly installed, 0 to remove and 88 not upgraded.
luhaozhhe@luhaozhhe-virtual-machine:~/Desktop/Data_Security/Lab1$ openssl enc -e -aes-128-cbc -in message.txt -out ciphertext.txt -K a3171d177d1ce97ebc644ea3ff826b4e -iv 8bc65f2f883f95eea10b6f940cc805f6 -base64
luhaozhhe@luhaozhhe-virtual-machine:~/Desktop/Data_Security/Lab1$ openssl enc -d -aes-128-cbc -in ciphertext.txt -out plaintext.txt -K a3171d177d1ce97ebc644ea3ff826b4e -iv 8bc65f2f883f95eea10b6f940cc805f6 -base64
luhaozhhe@luhaozhhe-virtual-machine:~/Desktop/Data_Security/Lab1$ g++ aes-128-cbc.cpp -o aes-128-cbc -lcrypto
luhaozhhe@luhaozhhe-virtual-machine:~/Desktop/Data_Security/Lab1$ ./aes-128-cbc
Hello World!
luhaozhhe@luhaozhhe-virtual-machine:~/Desktop/Data_Security/Lab1$
```

图 4.6: aes-128-cbc 加解密

我们发现，得到了加密后的密文，已经输出了正确的明文“Hello World!”，说明加解密程序是正确的。

#### 4.4 使用 OpenSSL 命令签名并验证

首先，生成 2048 位密钥，存储到文件 id\_rsa.key。

```
1 openssl genrsa -out id_rsa.key 2048
```

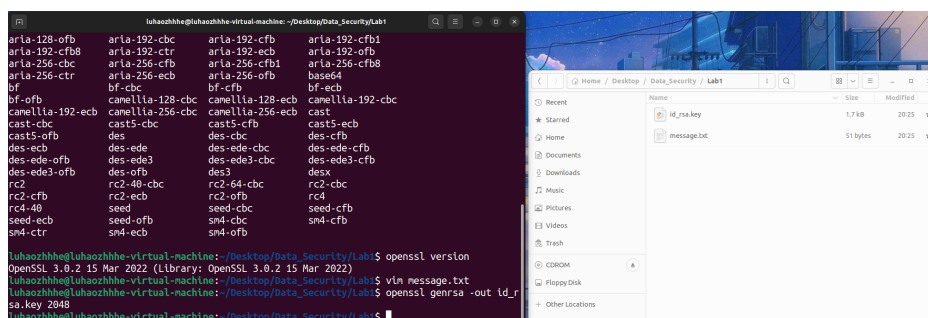


图 4.7: 生成 2048 位密钥

接着，根据私钥文件，导出公钥文件 id\_rsa.pub。

```
1 openssl rsa -in id_rsa.key -out id_rsa.pub -pubout
```

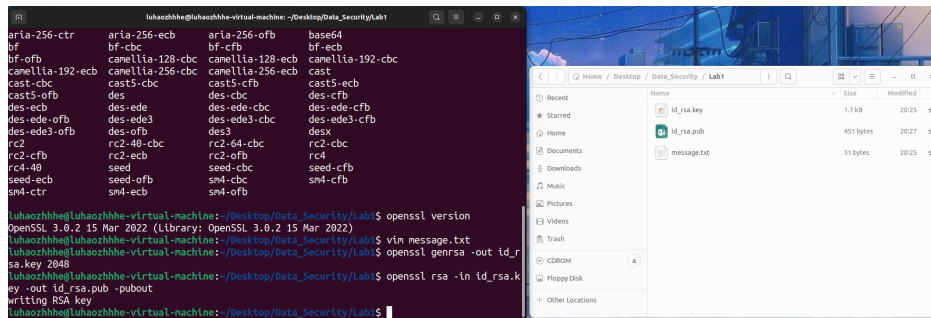


图 4.8: 导出公钥文件

使用私钥对文件 message.txt 进行签名，输出签名到 message.sha256。

```
1 openssl dgst -sign id_rsa.key -out rsa_signature.bin -sha256 message.txt
```

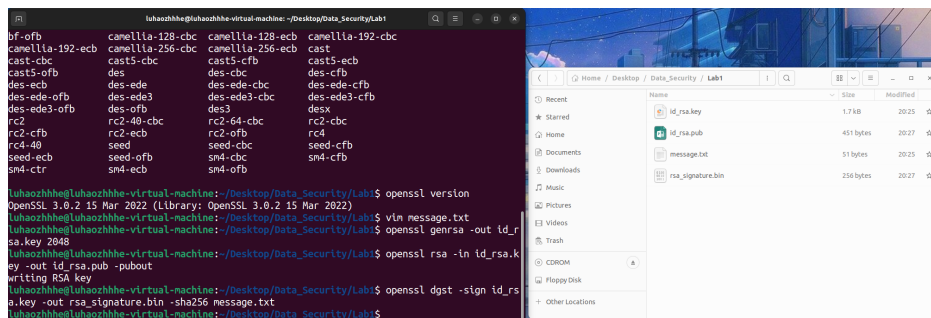


图 4.9: 使用私钥对文件 message.txt 进行签名

最后，使用公钥验证签名。

```
1 openssl dgst -verify id_rsa.pub -signature rsa_signature.bin -sha256 message.txt
```

可以看到输出 Verified OK，说明验证成功！

```
Luhaozhhe@Luhaozhhe-virtual-machine:~/Desktop/Data_Security/Lab1$ openssl dgst -verify id_rsa.pub -signature rsa_signature.bin -sha256 message.txt
Verified OK
```

图 4.10: 使用公钥验证签名

## 4.5 数字签名程序编写

课本上也给我们提供了对应的 cpp 源代码。

```
1 #include <stdio.h>
2 #include <string.h>
3 #include <openssl/evp.h>
4 #include <openssl/rsa.h>
```

```
5  #include <openssl/pem.h>
6  // 公钥文件名
7  #define PUBLIC_KEY_FILE_NAME "public.pem"
8  // 私钥文件名
9  #define PRIVATE_KEY_FILE_NAME "private.pem"
10 // RSA 生成公私钥，存储到文件
11 bool genrsa(int numbit)
12 {
13     EVP_PKEY_CTX *ctx = EVP_PKEY_CTX_new_id(EVP_PKEY_RSA, NULL);
14     if (!ctx)
15     {
16         return false;
17     }
18     EVP_PKEY *pkey = NULL;
19     bool ret = false;
20     int rt;
21     FILE *prif = NULL, *pubf = NULL;
22     if (EVP_PKEY_keygen_init(ctx) <= 0)
23     {
24         goto err;
25     }
26     // 设置密钥长度
27     if (EVP_PKEY_CTX_set_rsa_keygen_bits(ctx, numbit) <= 0)
28     {
29         goto err;
30     }
31     // 生成密钥
32     if (EVP_PKEY_keygen(ctx, &pkey) <= 0)
33     {
34         goto err;
35     }
36     prif = fopen(PRIVATE_KEY_FILE_NAME, "w");
37     if (!prif)
38     {
39         goto err;
40     }
41     // 输出私钥到文件
42     rt = PEM_write_PrivateKey(prif, pkey, NULL, NULL, 0, NULL, NULL);
43     fclose(prif);
44     if (rt <= 0)
45     {
46         goto err;
```

```
47     }
48     pubf = fopen(PUBLIC_KEY_FILE_NAME, "w");
49     if (!pubf)
50     {
51         goto err;
52     }
53     // 输出公钥到文件
54     rt = PEM_write_PUBKEY(pubf, pkey);
55     fclose(pubf);
56     if (rt <= 0)
57     {
58         goto err;
59     }
60     ret = true;
61 err:
62     EVP_PKEY_CTX_free(ctx);
63     return ret;
64 }
65 // 生成数据签名
66 bool gensign(const uint8_t *in, unsigned int in_len, uint8_t *out, unsigned int
    ↪ *out_len)
67 {
68     FILE *prif = fopen(PRIVATE_KEY_FILE_NAME, "r");
69     if (!prif)
70     {
71         return false;
72     }
73     // 读取私钥
74     EVP_PKEY *pkey = PEM_read_PrivateKey(prif, NULL, NULL, NULL);
75     fclose(prif);
76     if (!pkey)
77     {
78         return false;
79     }
80     bool ret = false;
81     EVP_MD_CTX *ctx = EVP_MD_CTX_new();
82     if (!ctx)
83     {
84         goto ctx_new_err;
85     }
86     // 初始化
87     if (EVP_SignInit(ctx, EVP_sha256()) <= 0)
```

```
88     {
89         goto sign_err;
90     }
91     // 输入消息, 计算摘要
92     if (EVP_SignUpdate(ctx, in, in_len) <= 0)
93     {
94         goto sign_err;
95     }
96     // 生成签名
97     if (EVP_SignFinal(ctx, out, out_len, pkey) <= 0)
98     {
99         goto sign_err;
100    }
101    ret = true;
102sign_err:
103    EVP_MD_CTX_free(ctx);
104ctx_new_err:
105    EVP_PKEY_free(pkey);
106    return ret;
107}
108// 使用公钥验证数字签名, 结构与签名相似
109bool verify(const uint8_t *msg, unsigned int msg_len, const uint8_t *sign,
110↪ unsigned int sign_len)
111{
112    FILE *pubf = fopen(PUBLIC_KEY_FILE_NAME, "r");
113    if (!pubf)
114    {
115        return false;
116    }
117    // 读取公钥
118    EVP_PKEY *pkey = PEM_read_PUBKEY(pubf, NULL, NULL, NULL);
119    fclose(pubf);
120    if (!pkey)
121    {
122        return false;
123    }
124    bool ret = false;
125    EVP_MD_CTX *ctx = EVP_MD_CTX_new();
126    if (!ctx)
127    {
128        goto ctx_new_err;
129    }
```

```
129 // 初始化
130 if (EVP_VerifyInit(ctx, EVP_sha256()) <= 0)
131 {
132     goto sign_err;
133 }
134 // 输入消息, 计算摘要
135 if (EVP_VerifyUpdate(ctx, msg, msg_len) <= 0)
136 {
137     goto sign_err;
138 }
139 // 验证签名
140 if (EVP_VerifyFinal(ctx, sign, sign_len, pkey) <= 0)
141 {
142     goto sign_err;
143 }
144 ret = true;
145 sign_err:
146     EVP_MD_CTX_free(ctx);
147 ctx_new_err:
148     EVP_PKEY_free(pkey);
149     return ret;
150 }
151 int main()
152 {
153     // 生成长度为 2048 的密钥
154     genrsa(2048);
155     const char *msg = "Hello World!";
156     const unsigned int msg_len = strlen(msg);
157     // 存储签名
158     uint8_t sign[256] = {0};
159     unsigned int sign_len = 0;
160     // 签名
161     if (!gensign((uint8_t *)msg, msg_len, sign, &sign_len))
162     {
163         printf(" 签名失败\n");
164         return 0;
165     }
166     // 验证签名
167     if (verify((uint8_t *)msg, msg_len, sign, sign_len))
168     {
169         printf(" 验证成功\n");
170     }
```

```
171     else
172     {
173         printf(" 验证失败\n");
174     }
175     return 0;
176 }
```

我们简单的进行分析：

1. 生成 RSA 公私钥对，并将其存储到文件中；
2. 使用私钥对数据进行签名；
3. 使用公钥验证数据的签名是否有效。

以下是各个函数的具体说明：

- `genrsa(int numbit)`: 该函数用于生成 RSA 公私钥对，并将其存储到文件中。首先创建一个 `EVP_PKEY_CTX` 对象，然后设置密钥长度，生成密钥，并分别将私钥和公钥输出到对应的文件中。
- `gensign(const uint8_t *in, unsigned int in_len, uint8_t *out, unsigned int *out_len)`: 该函数用于生成数据的签名。首先从私钥文件中读取私钥，然后初始化 `EVP_MD_CTX` 对象，计算消息摘要并生成签名。
- `verify(const uint8_t *msg, unsigned int msg_len, const uint8_t *sign, unsigned int sign_len)`: 这个函数用于使用公钥验证数据的签名是否有效。首先从公钥文件中读取公钥，然后初始化 `EVP_MD_CTX` 对象，计算消息摘要并验证签名的有效性。
- `main()`: 依次调用了上述函数，生成了长度为 2048 的 RSA 密钥对，对消息进行签名并验证签名的有效性。

在运行前，我们需要安装一些缺少的库，运行 `sudo apt-get install libssl-dev` 即可。

然后我们编译该代码，然后运行，得到以下结果：

```
1 g++ signature.cpp -o signature -lcrypto
2 ./signature
```

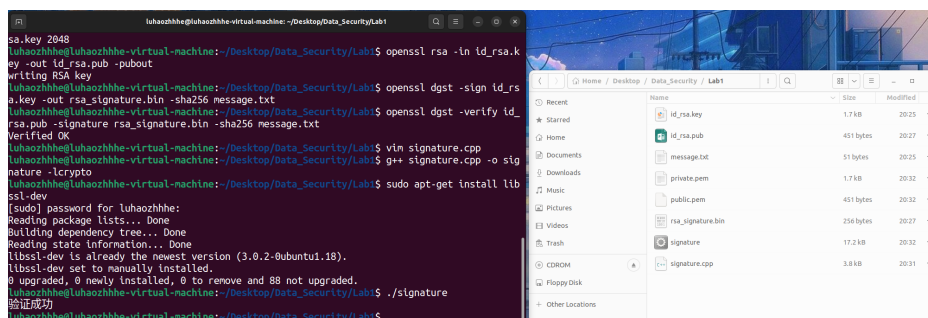


图 4.11: 编译并运行 signature.cpp

输出“验证成功”，说明程序正常运行。

至此，我们完成了本次作业的实验 2-2 的要求，附加上之前的实验 2-1，以及一些自己的探索过程。本次实验非常成功！

## 5 实验心得与体会

本次实验，我在我的 ubuntu 环境下，完成了该实验的基础要求以及一些课本上其他内容的探索。作为本学期数据安全的第一次实验，首先让我重新熟悉了 linux 的命令行使用，然后又复习了一下 OpenSSL 方面的知识。希望在本学期的学习中可以收获颇丰！