

南开大学

《数据安全》课程实验报告

实验一：数字签名应用实践



学 院 _____ 网络空间安全学院
专 业 _____ 信息安全
学 号 _____ 2112060
姓 名 _____ 孙璐

一、实验原理

1. RSA 算法

大整数因数分解问题是指：将两个大素数相乘十分容易，但想要对其乘积进行因数分解极其困难，因此可以将乘积公开作为加密密钥。

RSA 密码的公开加密密钥 $K_e(n, e)$ ，而保密的解密密钥 $K_d = \langle p, q, d, \Phi(n) \rangle$ ，保存 $p, q, \Phi(n)$ （欧拉函数，表示在比 n 小的正整数中与 n 互素的数的个数）是为了加速计算。

（1）密钥生成

- 随机选择两个大素数 p 和 q ， p 和 q 都保密；
- 计算 $n=pq$ ，将 n 公开；
- 计算 $\Phi(n)=(p-1)(q-1)$ ， $\Phi(n)$ 保密；
- 随机选取一个正整数 e ， $1 < e < \Phi(n)$ ，且 e 与 $\Phi(n)$ 互素， e 公开； e 和 n 就构成了用户的公钥；
- 根据 $ed \equiv 1 \pmod{\Phi(n)}$ ，计算出 d ， d 保密； d 和 n 构成了用户的私钥；

（2）加密： $C = M^e \bmod n$

（3）解密： $M = C^d \bmod n$

给定密文 $C = M^e \bmod n$ ，可以利用私钥 $K_d = \langle p, q, d, \Phi(n) \rangle$ 解密，即

$(C)^d = (M^e)^d = M^{ed} = M \bmod n$ ；通过 RSA 加解密运算可以看出，加密和解密运算具有可交换性： $D(E(m)) = (M^e)^d = M^{ed} = (M^d)^e = E(D(m)) \bmod n$

通过该可交换性看出，如果执行解密算法产生的签名，也可以通过公钥来进行验证。

2. OpenSSL

OpenSSL 是一个开源的安全套接字的密码库，包括常用的密码加解密算法，常用的密钥算法，证书管理和 SSL 协议等。

OpenSSL 提供了 8 种对称加密算法，包括 AES、DES、Blowfish、CAST、IDEA、RC2、RC5、RC4，4 种非对称加密算法，包括 DH 算法、RSA 算法、DSA 算法和椭

圆曲线算法 (EC)；实现了 5 种信息摘要算法，分别是 MD2、MD5、MDC2、SHA (SHA1) 和 RIPEMD。OpenSSL 还支持 SSL/TLS 协议，用于在网络通信中建立安全连接，以及进行证书管理等。

二、实验过程

1. Openssl 更新

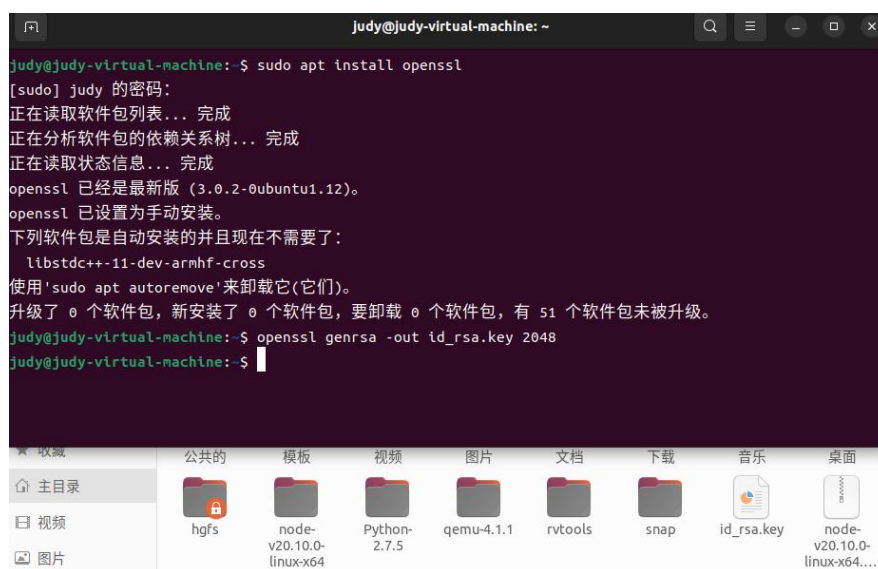
```
judy@judy-virtual-machine:~$ sudo apt install openssl
[sudo] judy 的密码:
正在读取软件包列表... 完成
正在分析软件包的依赖关系树... 完成
正在读取状态信息... 完成
openssl 已经是最新版 (3.0.2-0ubuntu1.10)。
openssl 已设置为手动安装。
升级了 0 个软件包，新安装了 0 个软件包，要卸载 0 个软件包，有 15 个软件包未被升级。
```

2. 在 openssl 中进行数据签名及验证

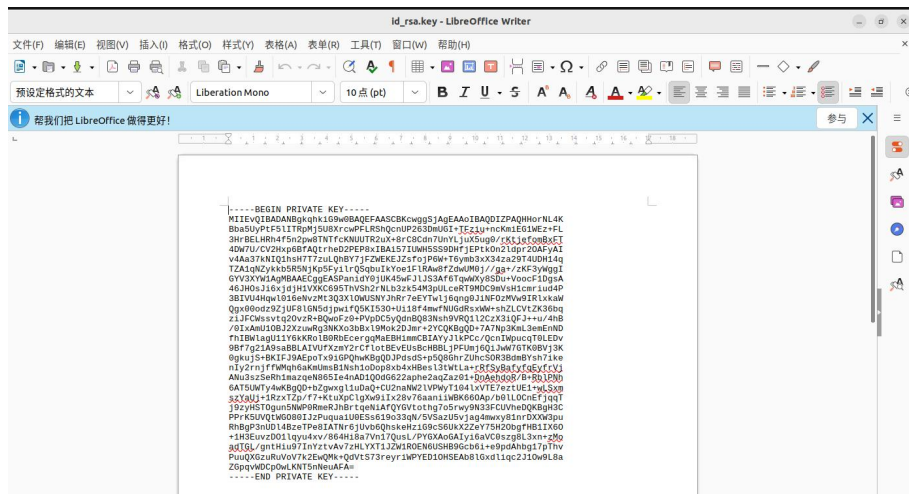
(1) 使用 openssl 命令签名并验证

① 生成 2048 位密钥，存储到公钥文件 id_rsa.key

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



```
judy@judy-virtual-machine:~$ sudo apt install openssl
[sudo] judy 的密码:
正在读取软件包列表... 完成
正在分析软件包的依赖关系树... 完成
正在读取状态信息... 完成
openssl 已经是最新版 (3.0.2-0ubuntu1.12)。
openssl 已设置为手动安装。
下列软件包是自动安装的并且现在不需要了:
  libstdc++-11-dev-armhf-cross
使用'sudo apt autoremove'来卸载它(它们)。
升级了 0 个软件包，新安装了 0 个软件包，要卸载 0 个软件包，有 51 个软件包未被升级。
judy@judy-virtual-machine:~$ openssl genrsa -out id_rsa.key 2048
judy@judy-virtual-machine:~$
```



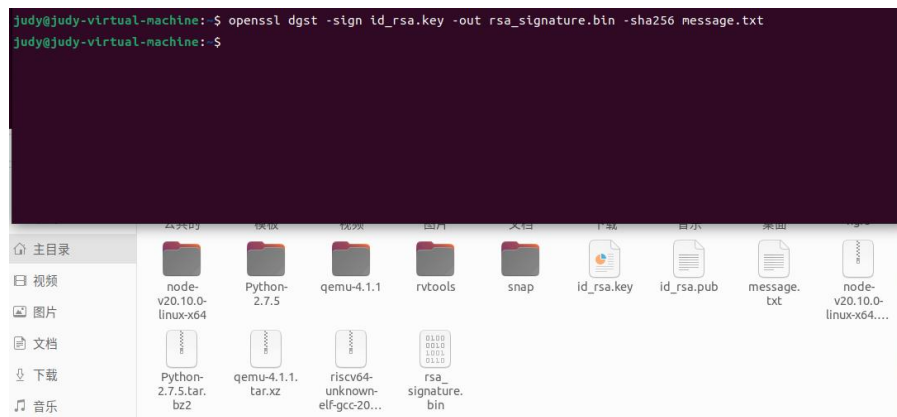
② 根据私钥文件，导出公钥文件 id_rsa.pub

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



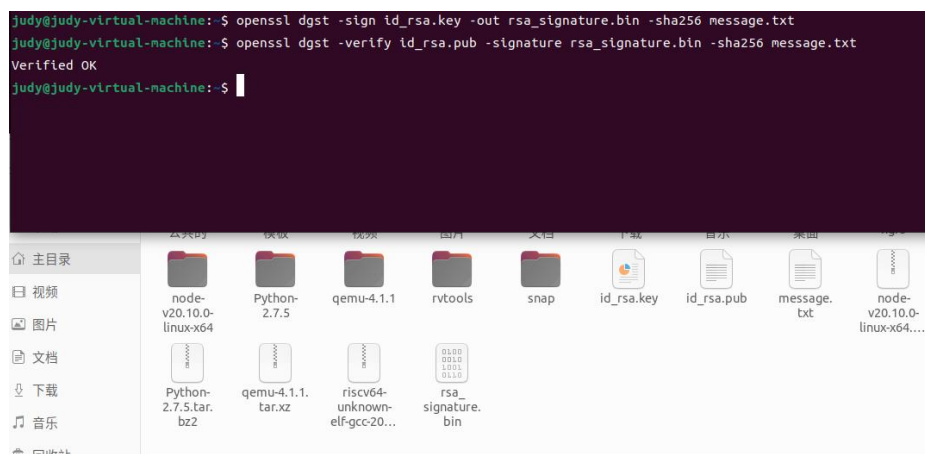
③ 使用私钥对文件 message.txt 进行签名，输出签名到 message.sha256

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



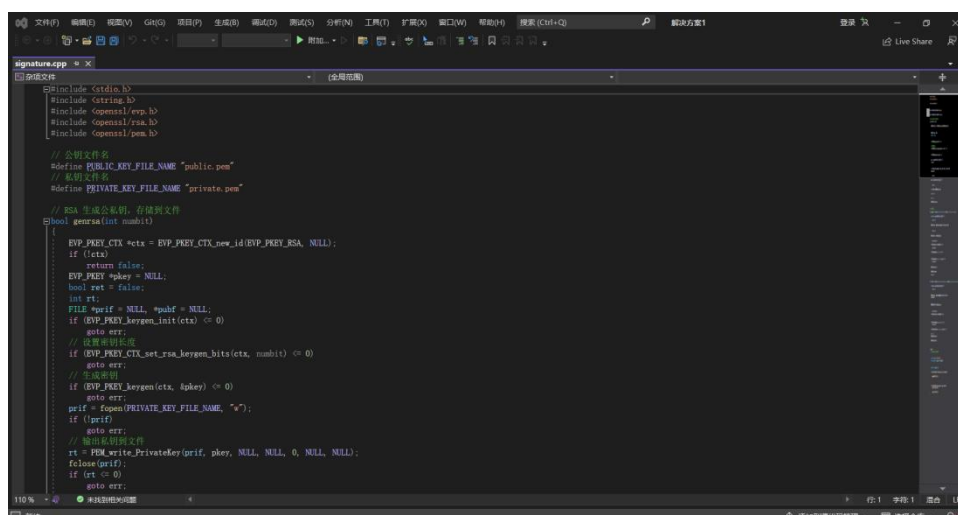
④ 使用公钥验证签名

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



3. 数字签名程序

① 按教材编写程序文件 `signature.cpp`



```
#include <stdio.h>
#include <string.h>
#include <openssl/evp.h>
#include <openssl/rsa.h>
#include <openssl/pem.h>

// 公钥文件名
#define PUBLIC_KEY_FILE_NAME "public.pem"
// 私钥文件名
#define PRIVATE_KEY_FILE_NAME "private.pem"

// RSA 生成公私钥，存储到文件
bool genrsa(int numbit)
{
    EVP_PKEY_CTX *ctx = EVP_PKEY_CTX_new_id(EVP_PKEY_RSA, NULL);
    if (!ctx)
        return false;
    EVP_PKEY *pkey = NULL;
    bool ret = false;
    int rt;
    FILE *prif = NULL, *pubf = NULL;
    if (EVP_PKEY_keygen_init(ctx) <= 0)
        goto err;
    // 设置密钥长度
    if (EVP_PKEY_CTX_set_rsa_keygen_bits(ctx, numbit) <= 0)
        goto err;
    // 生成密钥
    if (EVP_PKEY_keygen(ctx, &pkey) <= 0)
        goto err;
    prif = fopen(PRIVATE_KEY_FILE_NAME, "w");
    if (!prif)
        goto err;
    // 输出私钥到文件
    rt = PEM_write_PrivateKey(prif, pkey, NULL, NULL, 0, NULL, NULL);
    fclose(prif);
    if (rt <= 0)
        goto err;
}
```

```
#include <stdio.h>
#include <string.h>
#include <openssl/evp.h>
#include <openssl/rsa.h>
#include <openssl/pem.h>

// 公钥文件名
#define PUBLIC_KEY_FILE_NAME "public.pem"
// 私钥文件名
#define PRIVATE_KEY_FILE_NAME "private.pem"

// RSA 生成公私钥，存储到文件
bool genrsa(int numbit)
{
    EVP_PKEY_CTX *ctx = EVP_PKEY_CTX_new_id(EVP_PKEY_RSA, NULL);
    if (!ctx)
        return false;
    EVP_PKEY *pkey = NULL;
    bool ret = false;
    int rt;
    FILE *prif = NULL, *pubf = NULL;
    if (EVP_PKEY_keygen_init(ctx) <= 0)
        goto err;
    // 设置密钥长度
    if (EVP_PKEY_CTX_set_rsa_keygen_bits(ctx, numbit) <= 0)
        goto err;
    // 生成密钥
    if (EVP_PKEY_keygen(ctx, &pkey) <= 0)
        goto err;
    prif = fopen(PRIVATE_KEY_FILE_NAME, "w");
```

```

    if (!prif)
        goto err;
    // 输出私钥到文件
    rt = PEM_write_PrivateKey(prif, pkey, NULL, NULL, 0, NULL, NULL);
    fclose(prif);
    if (rt <= 0)
        goto err;

    pubf = fopen(PUBLIC_KEY_FILE_NAME, "w");
    if (!pubf)
        goto err;
    // 输出公钥到文件
    rt = PEM_write_PUBKEY(pubf, pkey);
    fclose(pubf);
    if (rt <= 0)
        goto err;
    ret = true;
err:
    EVP_PKEY_CTX_free(ctx);
    return ret;
}
// 生成数据签名
bool gensign(const uint8_t *in, unsigned int in_len, uint8_t *out, unsigned int *out_len)
{
    FILE *prif = fopen(PRIVATE_KEY_FILE_NAME, "r");
    if (!prif)
        return false;
    // 读取私钥
    EVP_PKEY *pkey = PEM_read_PrivateKey(prif, NULL, NULL, NULL);
    fclose(prif);
    if (!pkey)
        return false;
    bool ret = false;
    EVP_MD_CTX *ctx = EVP_MD_CTX_new();
    if (!ctx)
        goto ctx_new_err;
    // 初始化
    if (EVP_SignInit(ctx, EVP_sha256()) <= 0)
        goto sign_err;
    // 输入消息, 计算摘要
    if (EVP_SignUpdate(ctx, in, in_len) <= 0)
        goto sign_err;
    // 生成签名
    if (EVP_SignFinal(ctx, out, out_len, pkey) <= 0)

```

```

        goto sign_err;
    ret = true;
sign_err:
    EVP_MD_CTX_free(ctx);
ctx_new_err:
    EVP_PKEY_free(pkey);
    return ret;
}
// 使用公钥验证数字签名, 结构与签名相似
bool verify(const uint8_t *msg, unsigned int msg_len, const uint8_t *sign, unsigned int
sign_len)
{
    FILE *pubf = fopen(PUBLIC_KEY_FILE_NAME, "r");
    if (!pubf)
        return false;
    // 读取公钥
    EVP_PKEY *pkey = PEM_read_PUBKEY(pubf, NULL, NULL, NULL);
    fclose(pubf);
    if (!pkey)
        return false;
    bool ret = false;
    EVP_MD_CTX *ctx = EVP_MD_CTX_new();
    if (!ctx)
        goto ctx_new_err;
    // 初始化
    if (EVP_VerifyInit(ctx, EVP_sha256()) <= 0)
        goto sign_err;
    // 输入消息, 计算摘要
    if (EVP_VerifyUpdate(ctx, msg, msg_len) <= 0)
        goto sign_err;
    // 验证签名
    if (EVP_VerifyFinal(ctx, sign, sign_len, pkey) <= 0)
        goto sign_err;
    ret = true;
sign_err:
    EVP_MD_CTX_free(ctx);
ctx_new_err:
    EVP_PKEY_free(pkey);
    return ret;
}
int main()
{
    // 生成长度为 2048 的密钥
    genrsa(2048);

```



```

const char *msg = "Hello World!";
const unsigned int msg_len = strlen(msg);
// 存储签名
uint8_t sign[256] = {0};
unsigned int sign_len = 0;
// 签名
if (!gensign((uint8_t *)msg, msg_len, sign, &sign_len))
{
    printf("签名失败\n");
    return 0;
}
// 验证签名
if (verify((uint8_t *)msg, msg_len, sign, sign_len))
    printf("验证成功\n");
else
    printf("验证失败\n");
return 0;
}

```

这段代码引入了常用的库以及 OpenSSL 库，定义了公钥文件名为 "public.pem"，私钥文件名为 "private.pem"。

Genrsa 函数用于生成 RSA 公钥和私钥对。EVP_PKEY_CTX 结构用于密钥生成的上下文，EVP_PKEY_RSA 表示生成 RSA 密钥对。初始化密钥对生成操作的上下文，设置密钥长度 submit，EVP_PKEY_keygen(ctx, &pkey)生成密钥对，然后将公钥和私钥写入文件

gensign 函数用于对输入的数据进行签名。EVP_SignInit(ctx, EVP_sha256()) 初始化签名操作的上下文，使用 SHA256 哈希算法。EVP_SignUpdate(ctx, in, in_len)更新签名上下文，将要签名的数据加入到上下文中。最后 EVP_SignFinal(ctx, out, out_len, pkey)生成签名。

verify 函数用于验证签名是否有效。EVP_VerifyInit(ctx, EVP_sha256())：初始化验证操作的上下文，使用 SHA256 哈希算法。EVP_VerifyUpdate(ctx, msg, msg_len)更新验证上下文，将原始数据加入到上下文中。EVP_VerifyFinal(ctx, sign, sign_len, pkey)验证签名是否有效。

② 编译并运行

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



```
judy@judy-virtual-machine:~$ g++ signature.cpp -o signature -lcrypto
judy@judy-virtual-machine:~$ ./signature
验证成功
```

三、实验心得体会

本次实验中了解了 RSA 算法的原理和密钥生成过程。有助于更好地应用 RSA 算法进行数据加密和数字签名。通过实验中对 OpenSSL 库的使用，包括命令行工具和编程接口，掌握了一种常用的进行数字签名的工具和方法。了解了如何生成 RSA 密钥对、对数据进行签名和验证签名的过程，这为今后进行更复杂的加密操作打下了基础。数字签名可以确保数据的发送者和内容的完整性，有效防止了数据被篡改或伪造的风险，对于保障网络通信的安全至关重要。通过亲自动手进行数字签名实践，加深了对课堂所学知识的理解和掌握，使得抽象的概念变得更加具体和实用。