

**《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 孙蕗

1. **实验原理**
2. **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 ，而保密的解密密钥，保存 p,q,φ(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，φ(n)保密；

 · 随机选取一个正整数 e，1<e<φ(n)，且 e 与φ(n)互素，e 公开；e 和 n 就构成了用户的公钥；

 · 根据 ed≡1 mod φ(n)，计算出 d，d 保密；d 和 n 构成了用户的私钥；

（2）加密：

（3）解密：

给定密文，可以利用私钥 Kd=<p,q,d,φ(n)>解密，即；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：

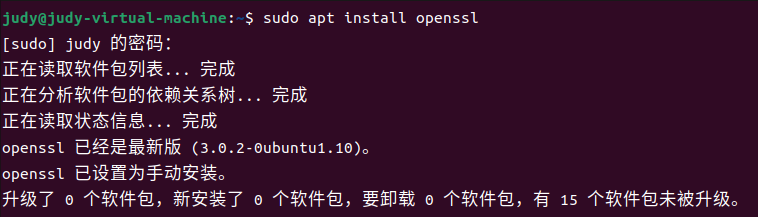
通过该可交换性看出，如果执行解密算法产生的签名，也可以通过公钥来进行验证。

1. **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。

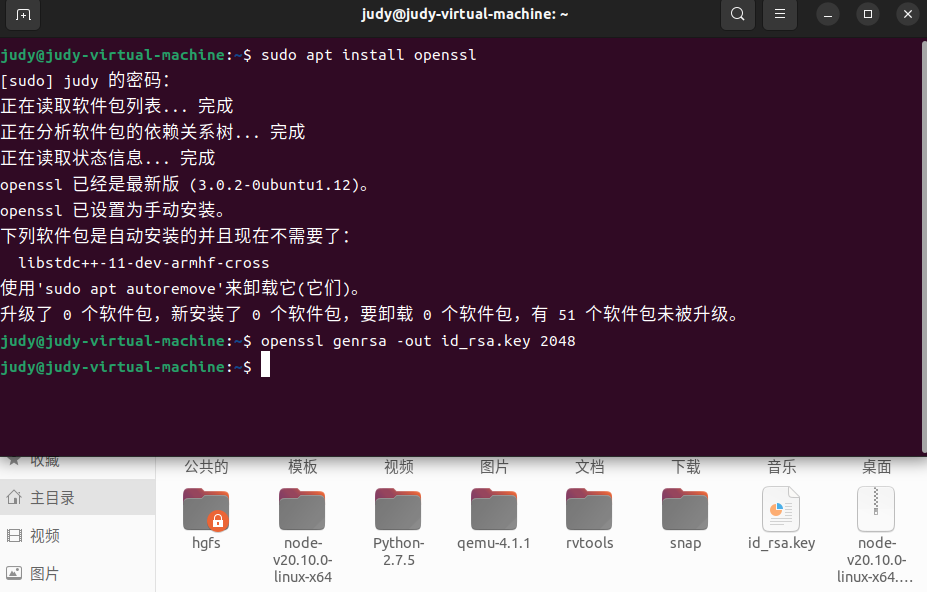
OpenSSL提供了8种对称加密算法，包括AES、DES、Blowfish、CAST、IDEA、RC2、RC5、RC4，4种非对称加密算法，包括DH算法、RSA算法、DSA算法和椭圆曲线算法（EC）；实现了5种信息摘要算法，分别是MD2、MD5、MDC2、SHA（SHA1）和RIPEMD。OpenSSL 还支持 SSL/TLS 协议，用于在网络通信中建立安全连接，以及进行证书管理等。

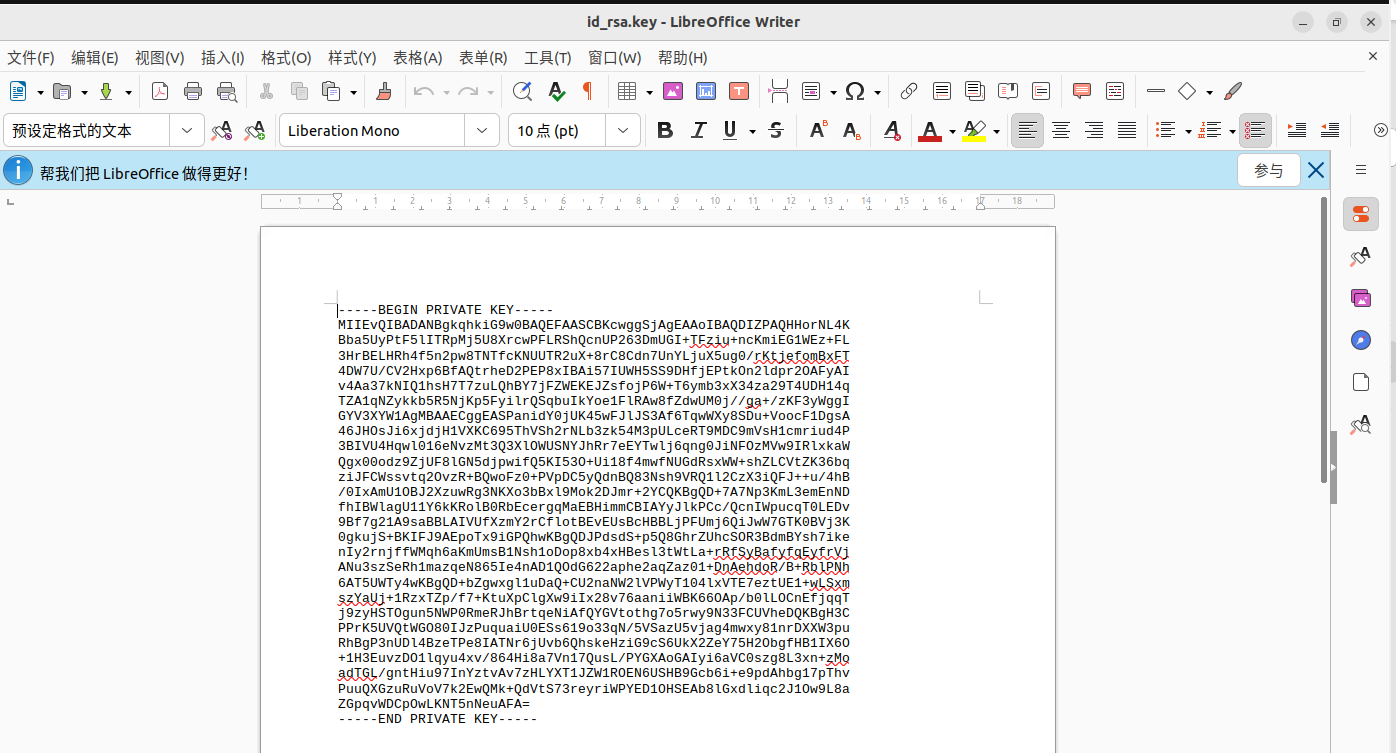
1. **实验过程**
2. Openssl更新



1. 在openssl中进行数据签名及验证
2. 使用openssl 命令签名并验证
3. 生成2048位密钥，存储到公钥文件id\_rsa.key

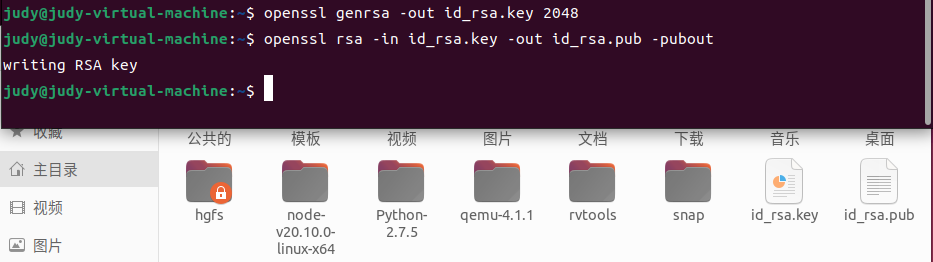
|  |
| --- |
| openssl genrsa -out id\_rsa.key 2048 |

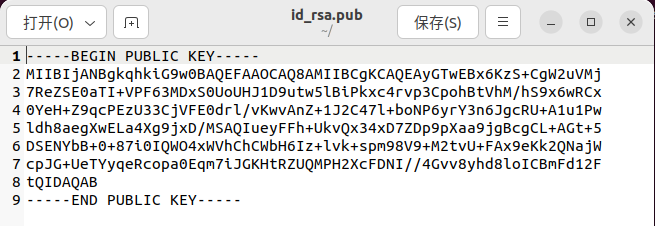




1. 根据私钥文件，导出公钥文件id\_rsa.pub

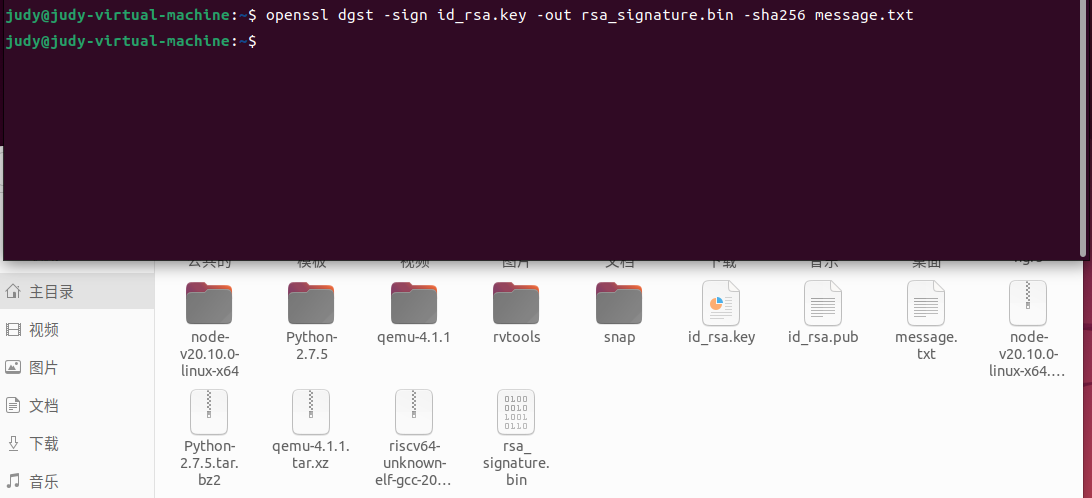
|  |
| --- |
| openssl rsa -in id\_rsa.key -out id\_rsa.pub -pubout |





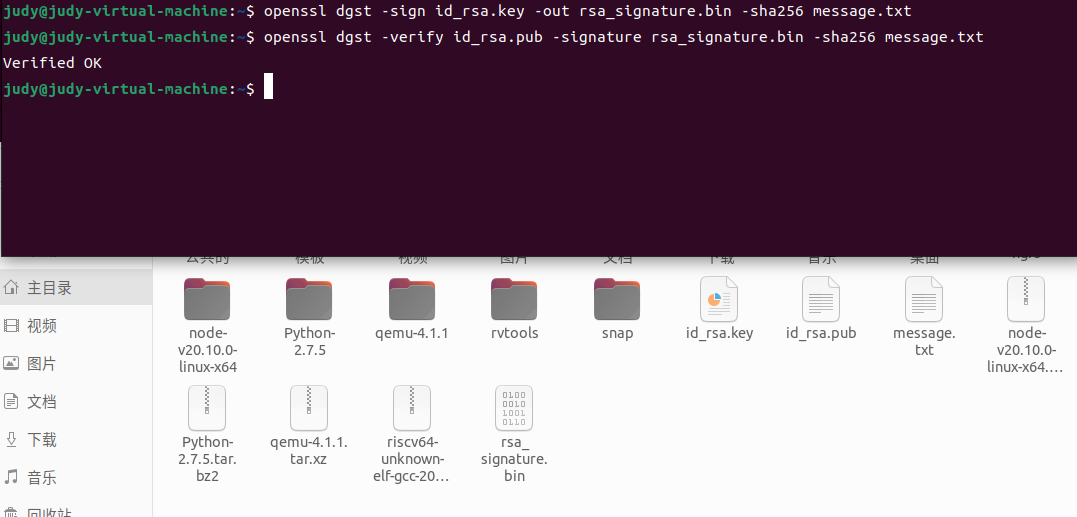
1. 使用私钥对文件message.txt进行签名，输出签名到message.sha256

|  |
| --- |
| openssl dgst -sign id\_rsa.key -out rsa\_signature.bin -sha256 message.txt |

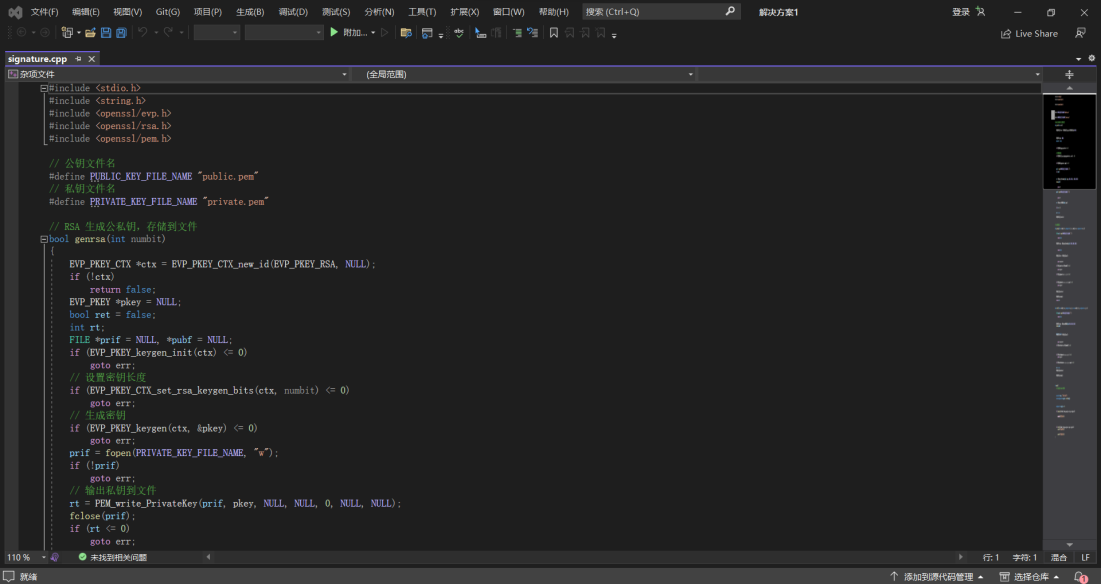


1. 使用公钥验证签名

|  |
| --- |
| openssl dgst -verify id\_rsa.pub -signature rsa\_signature.bin -sha256 message.txt |



1. 数字签名程序
2. 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;  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

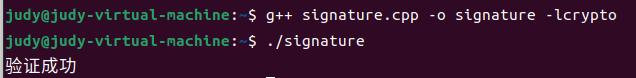
gensign 函数用于对输入的数据进行签名。EVP\_SignInit(ctx, EVP\_sha256())初始化签名操作的上下文，使用 SHA256 哈希算法。EVP\_SignUpdate(ctx, in, in\_len)更新签名上下文，将要签名的数据加入到上下文中。最后EVP\_SignFinal(ctx, out, out\_len, pkey)生成签名。

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。

1. 编译并运行

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





1. **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。