# 信息安全基础综合设计实验

***Lecture 06***

李经纬

电子科技大学

# 课程回顾

# 内容回顾

- OpenSSL密码学库
  - 安装：**`sudo apt-get install libssl-dev`**
- 大数运算
  - 数据类型：**`BIGNUM`**
  - 相关API函数：**`BN_new`、`BN_free`、`BN_dec2bn`、`BN_print_fp`**、...

# 模指数运算

➢**BN_mod_exp(BIGNUM *r, const BIGNUM *a, const BIGNUM *e, const BIGNUM *m, BN_CTX *ctx)**

- **计算辅助变量ctx**：使用BN_CTX_new()函数进行初始化

**string类的char*型成员变量**

```
BIGNUM *bn_a = BN_new(); // 初始化大数a, e, m, r等同理
BN_dec2bn(&bn_a, a.c_str()); // 将输入的字符串转换为大数
BN_CTX *ctx = BN_CTX_new(); // 初始化辅助变量ctx
BN_mod_exp(r, bn_a, bn_e, bn_m, ctx); // 计算模指数运算, r为计算结果
char *number_str = BN_bn2dec(r); // 将大数计算结果r转换为十进制字符串
BN_free(bn_a); // 手动释放大数变量, a,e,m,r同理
BN_CTX_free(ctx); // 手动释放辅助变量的内存空间
```

# 乘法逆元

➢**BIGNUM *BN_mod_inverse(BIGNUM *ret, const BIGNUM *a, const BIGNUM *m, BN_CTX *ctx)**

- OpenSSL各种变量的初始化和回收

```
BIGNUM *bn_a = BN_new(); // 初始化大数a; m, r等同理
BN_dec2bn(&bn_a, a.c_str()); // 将输入的字符串转换为大数
BN_CTX *ctx = BN_CTX_new(); // 初始化辅助变量ctx
BN_mod_inverse(r, bn_a, bn_m, ctx); // 计算模指数运算, r为计算结果
char *number_str = BN_bn2dec(r); // 将大数计算结果r转换为十进制字符串
BN_free(bn_a); // 手动释放大数变量, m、r同理
BN_CTX_free(ctx); // 手动释放辅助变量的内存空间
```

# RSA加密

# 加解密算法

- RSA三元组：(N, e, d )
  - 大素数：p和q
  - e和d为正整数，满足e*d mod (p-1)(q-1) = 1

- 加解密函数

| 公钥：**PK = (e, N)** | 私钥：**SK = (d, N)** |
| --- | --- |
| 加密函数：**E**(PK, M)<br>$C = M^e$ **mod** N | 解密函数：**D**(SK, C)<br>$M = C^d$ **mod** N |

# 算法实例

- 确定密钥
  - 假设选择 p = 2357和q = 2551为大素数，N = p*q = 6012707;
  - 选取 e = 422191，d = 3674911，满足 e*d **mod** (p-1)*(q-1) = 1
  - PK = (422191, 6012707)；SK = (3674911, 6012707)
- 加密
  - 编码后的消息 M = 5234673，计算 C = $5234673^{422191}$ mod 6012707 = 5411311
- 解密:
  - 计算 M = $5411311^{3674911}$ mod 6012707 = 5234673

# RSA密钥

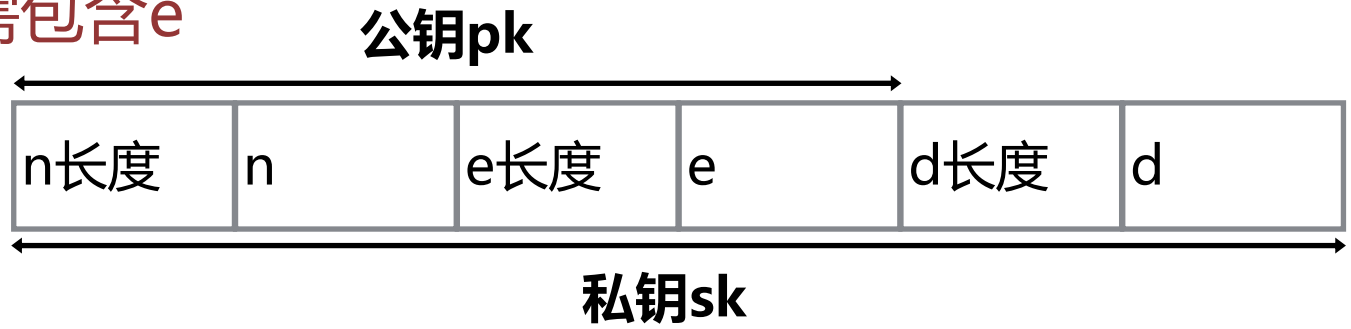
- 密钥生成：**RSA_generate_key**
  - 依赖头文件：`openssl/rsa.h`
  - 产生RSA结构体：`RSA *rsa = RSA_generate_key(1024, 3, NULL, NULL)`
    - 1024比特长度，初始e=3

**RSA结构体**

```
typedef struct {
    BIGNUM *n; // public modulus
    BIGNUM *e; // public exponent
    BIGNUM *d; // private exponent
    BIGNUM *p; // secret prime factor
    BIGNUM *q; // secret prime factor
    // ...
} RSA;
```

# 密钥序列化

- BN_bn2bin：转变为二进制保存
  - 注意: OpenSSL解密私钥需包含e




- 反序列化：BN_bin2bn

```
RSA* rsa = RSA_new();
memcpy(pk, 4, &len_n); // 前4个字节存储n的长度
BN_bin2bn(pk+4, len_n, rsa->n); // 接下来len_n个字节存储n的值
Memcpy(pk+4+len_n, 4, &len_e);
BN_bin2bn(pk+len_n+8, len_e, rsa->e);
```

# RSA加密

- **RSA_public_encrypt**

```
unsigned char* cipher = (unsigned char*)malloc(RSA_size(rsa));
int len_cipher = RSA_public_encrypt(strlen((char*)msg), msg,
cipher, rsa, RSA_PKCS1_OAEP_PADDING);
```

- **RSA_PKCS1_PADDING**：消息长度小于**RSA_size(rsa)-11**
- **RSA_PKCS1_OAEP_PADDING**：消息长度小于**RSA_size(rsa)-41**
- **RSA_NO_PADDING**：消息长度等于**RSA_size(rsa)**

# RSA解密

- **`RSA_private_decrypt`**
  - 解密函数Padding方式与加密函数保持一致
- **`$ man <openssl函数>`**
  - 查阅函数使用方法

# RSA命令行

```
// 生成RSA私钥，存入PEM文件
$ openssl genrsa -out rsa_pri.pem 2048

// 从RSA私钥中提取公钥，存入PEM文件
$ openssl rsa -in rsa_pri.pem -pubout -out rsa_pub.pem

// RSA加密
$ openssl rsautl -encrypt -in msg -inkey rsa_pub.pem -pubin -out
cipher

// RSA解密
$ openssl rsautl -decrypt -in cipher -inkey rsa_pri.pem -out msg
```

# RSA签名

# 数字签名

- 现实生活中的签名：签字、盖章、指纹...
  - 签名具有法律效力
- 数字环境中的签名：**数字签名，一种数学变换**
  - 完整性
  - 不可伪造性
  - 不可抵赖性

# RSA签名算法

- RSA三元组：(N, e, d )
  - 大素数：p和q
  - e和d为正整数，满足e*d mod (p-1)(q-1) = 1

- 签名&验证函数

| 公钥：PK = (e, N) | 私钥：SK = (d, N) |
| --- | --- |
| 签名函数：**S**(SK, M)<br>$s = M^d$ **mod** N | 验证函数：**V**(PK, s, M)<br>**if** $s^e$ **mod** N = M<br>签名合法<br>**end if** |

# OpenSSL实现

- 函数：**RSA_sign & RSA_verify或RSA_private_encrypt & RSA_public_decrypt**
  - 可通过**man命令（或OpenSSL手册）**查看函数用法

- 命令行：

```
// 签名
$ openssl dgst -sign private.pem -sha256 -out sign.txt file.txt
// 验证
$ openssl dgst -verify pub.pem -sha256 -signature sign.txt file.txt
```

# 课堂作业

# RSA密钥文件读取

- 打开密钥文件<filename>

```
FILE *fp = NULL; // 初始化文件指针
if ((fp = fopen(<filename>, "r")) == NULL){ // 判断文件是否能正常打开
        // 错误处理，条件中等于NULL说明文件打开错误，反之代表文件正常打开
}
……
fclose(fp); // 对文件的操作完成后,关闭文件(即完成公私钥读取后)
```

- 读取公私钥
  - **PEM_read_RSA_PUBKEY**
  - **PEM_read_RSAPrivateKey**

# RSA密钥载入

- 载入PEM文件以初始化RSA密钥

```
RSA *rsa_private_key = NULL; // rsa私钥
RSA *rsa_public_key = NULL; // rsa公钥
bool load_RSA_keys() {
//  返回值:
//      true 代表读取成功
//      false 代表读取失败
// 其他说明:
//      通过宏定义的路径将rsa公钥、私钥分别读取到rsa_private_key和
rsa_public_key
}
```

# RSA加密与解密

- 密钥载入完毕后，完成RSA加解密操作
- 函数头：

```
string RSA_Encryption(string plaintext) {
    load_RSA_keys();
//  参数：plaintext 代表输入的明文字符串
//  返回值：string类型，返回加密结果
}

string RSA_Decryption(string ciphertext)
    load_RSA_keys();
//  参数：ciphertext 代表输入的密文字符串
//  返回值：string类型，返回解密结果
}
```

# RSA签名与验证

- 密钥载入完毕后，完成RSA加解密操作
- 函数头：

```
string RSA_signature_signing(string input) {
    load_RSA_keys();
//  参数: input 代表要签名的明文字符串
//  返回值: string类型, 返回签名的结果
}

bool RSA_signature_verify(string message, string
signature) {
    load_RSA_keys();
//  参数: message 代表输入的签名结果；signature 代表签名的结果
//  返回值: bool类型, 成功返回true, 失败返回false
}
```