

内容回顾

❖ 随机数与伪随机数

→ 伪随机数: 通过算法产生, 近似随机数, 可以重现

❖ 线性同余伪随机数生成器

→ 迭代: $X_{i+1} = aX_i + c \bmod m$

→ NIST伪随机数判断标准

❖ BBS伪随机数生成器

→ 参数选取: 素数 p 和 q ($p \bmod 4 = q \bmod 4 = 3$), $N = p * q$

→ 迭代: $X_{i+1} = X_i^2 \bmod N$, 选取 X_{i+1} 的重要比特位

→ 安全性: 可证明安全, 规约为大数难分解问题

作业讲评: 线性同余算法

- ❖ 已知 X_i , 产生下一个伪随机数

```
1: return (a*Xi + c) mod m
```

- ❖ 控制迭代次数: 递归

```
1: if n > 0  
2:   output X  
3:   RandLCG(a*X+c, m, a, c, n-1)  
4: end if
```

作业讲评-BBS算法

❖ 迭代: 递归函数

1: if $n > 0$ 2: $s = s^2 \bmod p \cdot q$ 3: 选择并输出 s 的重要位	4: RandBBS ($s, p, q, n-1$) 5: end if
---	---

❖ 选择重要位: 令 $v = s$

最低位

$b = v \& 1$

奇校验位

```
b = 1
while v is not 0
  if v & 1 == 1
    b = !b
  end if
  v = v >> 1
end while
```

偶校验位

```
b = 0
while v is not 0
  if v & 1 == 1
    b = !b
  end if
  v = v >> 1
end while
```

ANSI X9.17生成器

❖ Triple DES (3DES)

- ➔ 对称加密算法: $K = (K_1, K_2, K_3)$
- ➔ 加密函数 **F**: $\mathbf{E}(K_3, \mathbf{D}(K_2, \mathbf{E}(K_1, M)))$, **E**和**D**为DES加解密函数

❖ 算法流程

- ➔ 输入: 64比特种子 s_0 , 3DES密钥 K
- ➔ 校正: 加密当前系统时间 $I = \mathbf{F}(K, \text{Time})$
- ➔ 迭代: $X_i = \mathbf{F}(K, I \mathbf{XOR} s_i), s_{i+1} = \mathbf{F}(K, X_i \mathbf{XOR} I)$
- ➔ 输出: $\{X_i\}$

基于RSA的非对称密码实验

基于OpenSSL的大数运算库

RSA加密和签名

非对称密码

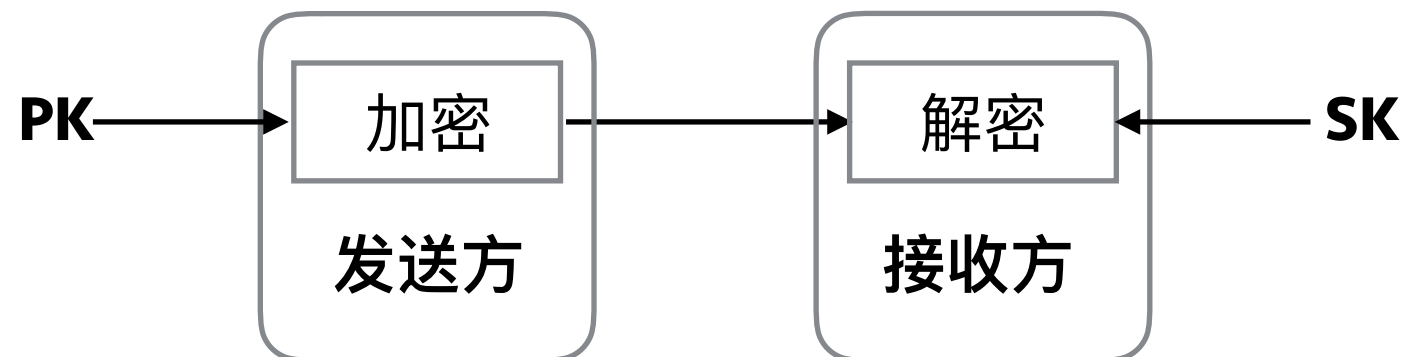
❖ 对称加密的缺陷: 发送方和接收方共享对称密钥

➔ 需密钥协商

❖ 1976年论文“**New Directions in Cryptography**”提出了**非对称密码 (公钥密码)**

➔ 密钥由一对公私钥(PK, SK)组成, PK用于加密, SK用于解密

发送方使用接收方的**PK**加密, 接收方使用自己的**SK**解密



大数运算

- ❖ 基于**RSA**非对称密码实现
 - ➔ 其他实现方式: ECC, 格...
- ❖ RSA基础: **大数运算**
 - ➔ 基本数据类型表示的数值范围有限, 难以满足大规模数值计算
- ❖ 实现: **实现BIGNUM**

OpenSSL安装和使用

- ❖ **OpenSSL**是一个**开源**的密码学库: 包含大数运算库
- ❖ 安装: **sudo apt-get install libssl-dev**
 - ➔ **sudo**: 以管理员身份执行命令
- ❖ 使用: **g++ -o {可执行文件} {源程序} [-I库头文件所在目录] [-L库所在目录] -lcrypto**
 - ➔ 头文件和库文件已在环境变量**LD_LIBRARY_PATH**和**CPLUS_INCLUDE_PATH(C_INCLUDE_PATH)**的搜索路径中

BIGNUM库-基础

❖ 头文件: **#include <openssl/bn.h>**

❖ BIGNUM基础: 初始化和回收

➔ 初始化: BN_init, BN_new

➔ 回收: BN_free

1: BIGNUM static_bn, *dynamic_bn;

2: **BN_init**(&static_bn);

// 初始化静态BIGNUM结构

3: dynamic_bn = **BN_new**();

// 动态分配BIGNUM结构的内存

4: **BN_free**(dynamic_bn);

// 内存回收

BIGNUM库-赋值和输出

❖ 赋值函数: **BN_zero**, **BN_one**, **BN_set_word**

❖ 输出函数: **BN_print_fp**

1: BIGNUM *bn;	7: BN_one (bn);
2: unsigned char *buffer;	8: BN_print_fp (stdout, bn);
3: bn = BN_new();	9: BN_set_word (bn, 1024);
	10: BN_print_fp (stdout, bn);
4: BN_zero (bn);	
5: BN_print_fp (stdout, bn);	11: BN_free(bn);

BIGNUM库-存储

❖ 二进制转换

```
1: BIGNUM *num;  
2: len = BN_num_bytes(num)  
3: buf = (unsigned char *)malloc(len * sizeof(unsigned char));  
4: len = BN_bn2bin(num, buf);
```

编程练习

1. 初始化BIGNUM结构bn
2. 为bn赋值1024
3. 将bn输出为二进制buffer, 观察buffer存储表示

BIGNUM的二进制形式以**大根(big-endian)**形式存储

BIGNUM库-算术运算

❖ BIGNUM API

❖ 以模指数运算为例: **BN_mod_exp**

```
1: BIGNUM *r, *a, *e, *m;  
2: BN_CTX *ctx = BN_CTX_new();    // 初始化用于存储临时信息的环境  
   /* .. call BN_new() on r, g, x, p */  
   /*.. make assignments on a, e, m */  
3: BN_mod_exp(r, a, e, m, ctx);  
   /* .. call BN_free() on r, g, x, p */
```

➡ a, e, m可为**任意**二进制长度

BIGNUM库-算术运算

❖ BIGNUM API

❖ 以乘法逆元为例: **BN_mod_inverse**

```
1: BIGNUM *r, *a, *m;  
2: BN_CTX *ctx = BN_CTX_new();    // 初始化用于存储临时信息的环境  
   /* .. call BN_new() on r, g, x, p */  
   /*.. make assignments on a, e, m */  
3: BN_mod_inverse(r, a, m, ctx);  
   /* .. call BN_free() on r, g, x, p */
```

➔ a, m可为**任意**二进制长度

BIGNUM库-素数

- ❖ **BN_generate_prime_ex**(BIGNUM *ret, int bits, int safe, NULL, NULL, NULL)
 - ➔ ret: 产生的伪随机素数
 - ➔ bits: 产生的素数的最小长度
 - ➔ safe: 是否为安全素数 ((ret-1)/2也为素数)
- ❖ 素性判定函数: **BN_is_prime_ex**

BIGNUM库-伪随机数

- ❖ **BN_random**(BIGNUM *rnd, int bits, int top, int bottom)
 - ➔ top: BN RAND TOP ANY, BN RAND TOP ONE, BN RAND TOP TWO
 - ➔ bottom: BN RAND BOTTOM ODD, BN RAND BOTTOM ANY
- ❖ **BN_rand_range**(BIGNUM *rnd, BIGNUM *range)
 - ➔ range: 产生0~range的伪随机数
- ❖ 参考[更多基于OpenSSL的大数操作](#)

基于RSA的非对称密码实验

基于OpenSSL的大数运算库

RSA加密和签名

RSA加解密算法

❖ RSA三元组: **(e, d, N)**

→ 大素数p和q, $N = p * q$

→ e和d为正整数, 满足 $e * d \bmod (p-1)(q-1) = 1$

❖ 加解密函数

公钥: PK = (e, N)	私钥: SK = (d, N)
加密函数: E(PK, M) $C = M^e \bmod N$	解密函数: D(SK, C) $M = C^d \bmod N$

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**

→ **完整性**

→ **不可伪造性**

→ **不可抵赖性**

数字签名

数字签名是一种**密码变换**, 用来保护数据完整性, 不可伪造性和不可抵赖性

基于RSA的数字签名算法

❖ RSA三元组: **(e, d, N)**

→ 大素数p和q, $N = p * q$

→ e和d为正整数, 满足 $e * d \bmod (p-1)(q-1) = 1$

❖ 数字签名算法

公钥: $PK = (e, N)$	私钥: $SK = (d, N)$
签名函数: S (SK, M) $s = M^d \bmod N$	验证函数: V (PK, s, M) if $s^e \bmod N = M$ 签名合法 end if

基于RSA加密算法实例

❖ 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 $e*d \bmod (p-1)*(q-1) = 1$

➔ $PK = (422191, 6012707)$; $SK = (3674911, 6012707)$

❖ 加密: 编码后的消息 $M = 5234673$, 计算 $C = 5234673^{422191} \bmod 6012707 = 5411311$

❖ 解密: 计算 $M = 5411311^{3674911} \bmod 6012707 = 5234673$

OpenSSL RSA-密钥

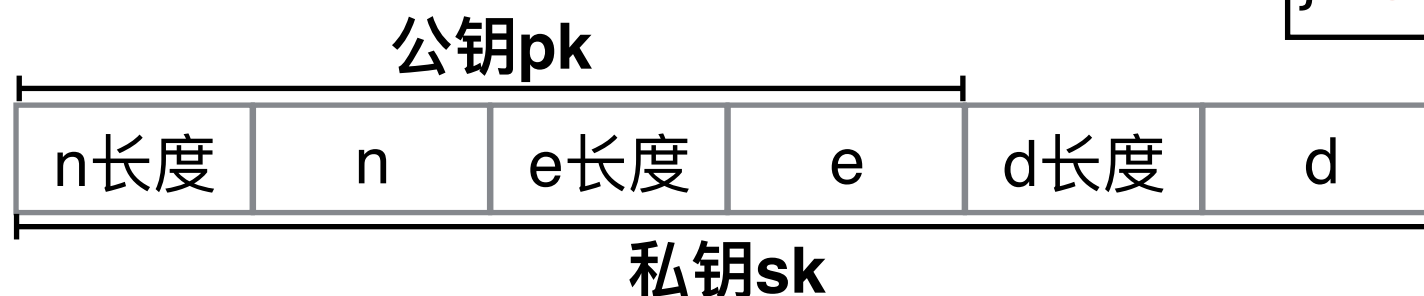
❖ Alice和Bob基于RSA保密通信

❖ 产生Bob的RSA密钥: **RSA_generate_key**

➔ RSA密钥: 1024比特长度, $e = 3$

```
1: #include <openssl/rsa.h>
2: RSA *rsa = RSA_generate_key(1024,
                               3, NULL, NULL);
```

❖ 公钥/私钥序列化: **BN_bn2bin**



RSA结构体

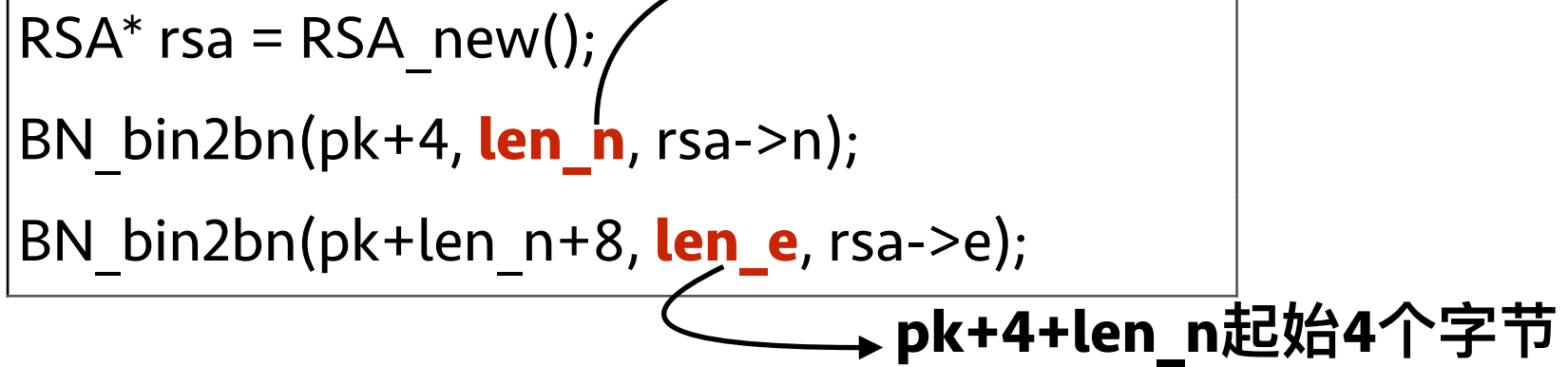
```
typedef struct {
    BIGNUM *n; // public modulus
    BIGNUM *e; // public exponent
    BIGNUM *d; // private exponent
    BIGNUM *p; // secret prime factor
    BIGNUM *q; // secret prime factor
    // ...
} RSA;
```

注意: OpenSSL解密私钥需包含e

OpenSSL RSA-加密

❖ 公钥解析: **BN_bin2bn**

```
RSA* rsa = RSA_new();  
BN_bin2bn(pk+4, len_n, rsa->n);  
BN_bin2bn(pk+len_n+8, len_e, rsa->e);
```



pk起始4个字节

pk+4+len_n起始4个字节

❖ Alice加密消息

```
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**: 明文消息长度小于RSA_size(rsa)-11

→ **RSA_PKCS1_OAEP_PADDING**: 明文消息长度小于RSA_size(rsa)-11

OpenSSL RSA-解密

❖ 私钥解析: **BN_bin2bn**

```
RSA* rsa = RSA_new();  
BN_bin2bn(pk+4, len_n, rsa->n);  
BN_bin2bn(pk+len_n+8, len_e, rsa->e);  
BN_bin2bn(pk+len_n+len_e+12, len_d, rsa->d);
```

❖ Bob解密消息

```
int len_msg = RSA_private_decrypt(len_cipher, cipher, msg, rsa,  
                                   RSA_PKCS1_OAEP_PADDING);
```

OpenSSL RSA-其他

❖ 基于命令行的RSA密钥生成

➔ 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
```

❖ 基于命令行的RSA加密和解密

\$ openssl rsautl -encrypt -in msg -inkey rsa_pub.pem -pubin -out cipher	加密
\$ openssl rsautl -decrypt -in cipher -inkey rsa_pri.pem -out msg	解密

OpenSSL RSA-签名和验证

❖ 签名: **RSA_private_encrypt**

→ 产生消息msg的一次解密结果, 作为签名sig

❖ 验证: **RSA_public_decrypt**

→ 加密sig, 验证加密密文是否等于msg