信息安全基础综合设计实验

Lecture 07

李经纬

电子科技大学

课程回顾

RSA加密&签名

- 参数选择：p和q为大素数，$N = p*q$，$e*d \bmod (p-1)(q-1) = 1$
  - PK = (N, e)；SK = (N, d)
- 加密：$C = M^e \bmod N$；$M = C^d \bmod N$
- 签名&验证：$s = M^d \bmod N$；判断$s^e \bmod N \stackrel{?}{=} M$
  - 签名具有：**完整性、不可伪造性、不可抵赖性**
- 安全性：归约到**大数难分解困难问题**

作业概览

- 密钥载入
  - 通过文件指针**FILE *fp**打开密钥文件（fopen）
  - 读取PEM格式公钥（PEM_read_RSA_PUBKEY）
    - 读取PKCS＃1格式公钥（PEM_read_RSAPublicKey）
  - 读取私钥（PEM_read_RSAPrivateKey）
- 加密（RSA_public_encrypt）；解密（RSA_private_decrypt）
- 签名（RSA_private_encrypt）；验证（RSA_public_decrypt）

密钥载入

- 以载入公钥为例

```
FILE *fp = NULL; //定义文件指针
if ((fp = fopen(PUBLICKEY, "r")) == NULL) { // 判断公钥文件是否正确打开
    … // 错误处理
}
if ((rsa_private_key = PEM_read_RSA_PUBKEY(fp, NULL, NULL, NULL))
== NULL) { // 通过函数读取公钥，并判断是否成功
    … // 错误处理
}
fclose(fp); // 读取成功，关闭已经打开的公钥文件
```

加密

- 要点：根据padding方式，正确处理输入数据长度
  - 当消息长度超过处理长度限制时，须进行分块
  - 加密结果长度为RSA密钥长度（**RSA_size**）

```
int rsa_len = RSA_size(rsa_public_key); // 获取密钥长度
unsigned char *encryptMsg = (unsigned char *)malloc(rsa_len);
memset(encryptMsg, 0, rsa_len); // 使用0填充encryptMsg指向的一段内存空间
if (RSA_public_encrypt(input.length(), (unsigned char *)input.c_str(),
encryptMsg, rsa_public_key, RSA_PKCS1_PADDING) >= 0) {
    string returnVal((char *) encryptMsg, rsa_len); // 隐式调用string
类构造函数（注：encryptMsg可能包含特殊字符，导致部分赋值）
}
```

解密

```
int rsa_len = RSA_size(rsa_private_key); // 获取RSA密钥长度(Byte为单位)

unsigned char *decryptMsg = (unsigned char *)malloc(rsa_len);
memset(decryptMsg, 0, rsa_len);

// 解密padding方式与加密的padding方式一致
if (RSA_private_decrypt(rsa_len, (unsigned char *)input.c_str(),
decryptMsg, rsa_private_key, RSA_PKCS1_PADDING) >= 0) {

    // 隐式调用string类构造函数（注：decryptMsg符合自然语义，不包含特殊字符）
    string returnVal((char *) decryptMsg);
}
```

签名

- 调用RSA_private_encrypt底层函数

```
int rsa_len = RSA_size(rsa_private_key);

unsigned char *encryptMsg = (unsigned char *)malloc(rsa_len);
memset(encryptMsg, 0, rsa_len);

// 签名使用私钥
if (RSA_private_encrypt(input.length(), (unsigned char *)input.c_str(),
encryptMsg, rsa_private_key, RSA_PKCS1_PADDING) >= 0) {
    string returnVal((char *) encryptMsg, rsa_len);
}
```

验证

- 调用RSA_public_decrypt函数，并与原始消息比较

```
int rsa_len = RSA_size(rsa_public_key);
unsigned char *decryptMsg = (unsigned char *)malloc(rsa_len);
memset(decryptMsg, 0, rsa_len);
if (RSA_public_decrypt(rsa_len, (unsigned char *)input.c_str(),
decryptMsg, rsa_public_key, RSA_PKCS1_PADDING) >= 0) {
    string decryptVal((char *) decryptMsg);
    // 与原消息比较
    if (message == decryptVal)
        return true;
    else
        return false;
}
```

流密码：RC4

对称密码

- 加解密使用**相同**密钥
  - 对比：非对称加密密钥分为公钥和私钥
- 解决安全通信问题
  - 优点：加解密速度快，密钥管理简单，适合一对一通信
  - 缺点：密钥分发困难，不适合一对多加密传输

流密码历史

- 一次一密：E(K, M) = K XOR M
  - 完美隐私性（perfect secrecy）
  - 问题：|K| = |M|
- 流密码：使用伪随机数代替K
  - 伪随机数生成器G：$\{0, 1\}^k \rightarrow \{0, 1\}^*$
    - **k为有限长度**，G()可为任意长度
  - E(K, M) = G(K) XOR M

流密码

- 一种**对称加密**算法，加解密双方（基于密钥）产生相同伪随机流，明文与伪随机流**按位**异或加密
  - 加密单位：比特位
- 优点：低错误传播，硬件实现简单
  - 适用于较高传输错误的通信环境
- 缺点：扩散度低

RC4原理

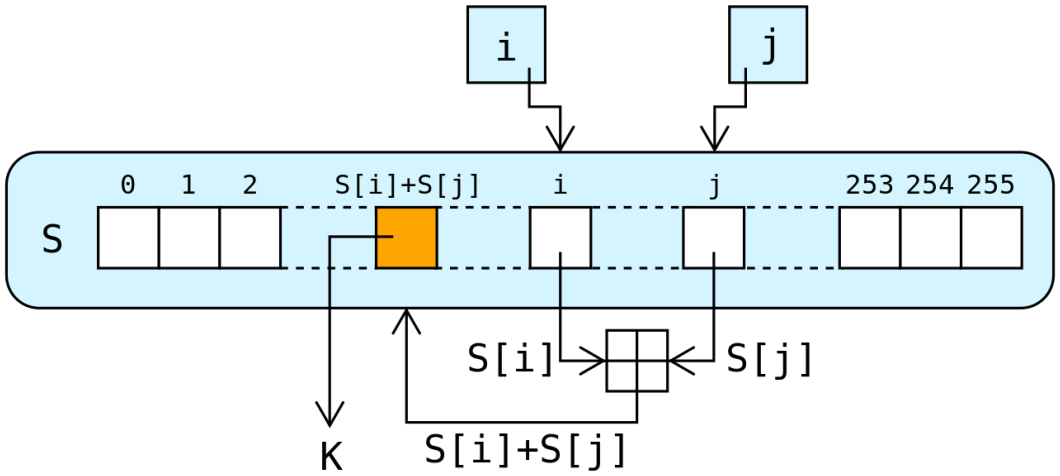
- RC4：一种具有**可变密钥长度（1~255字节）**的流密码
- 基于256字节**状态数组**（初始化为单位数组）
  - **KSA算法**：基于K置换状态数组
  - **PRNG算法**：扩充状态数组，加密明文数据

KSA算法

```
// 初始状态数组为单位数组
for (int i = 0; i < 256; i++) {
    S[i] = i;
}

// 构造S的置换
j = 0;
for (int i = 0; i < 256; i++) {
    j = j + S[i] + K[i mod len(K)] mod 256;
    swap values of S[i] and S[j];
}
```

PRNG算法

```
i = 0, j = 0;
for (int i = 0; i < len(M); i++) {
    i = i + 1 mod 256;
    j = j + S[i] mod 256;
    swap values of S[i] and S[j];
    C[k] = M[k] XOR S[S[i]+S[j]];
}
```



OpenSSL中的RC4

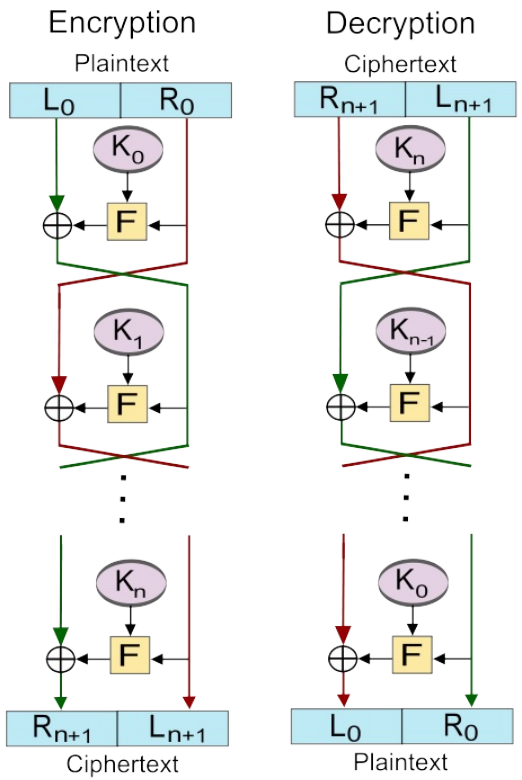
- 头文件依赖：**openssl/rc4.h**
- 产生状态数组的置换：**RC4_set_key(RC4_KEY *key, int len, const unsigned char *data)**
  - **注：key-状态数组；len-密钥长度；data-密钥**
- 数据加密&解密：**RC4(RC4_KEY *key, unsigned long len, const unsigned char *indata, unsigned char *outdata)**
  - **注：len-数据长度**

分组密码：DES

分组密码

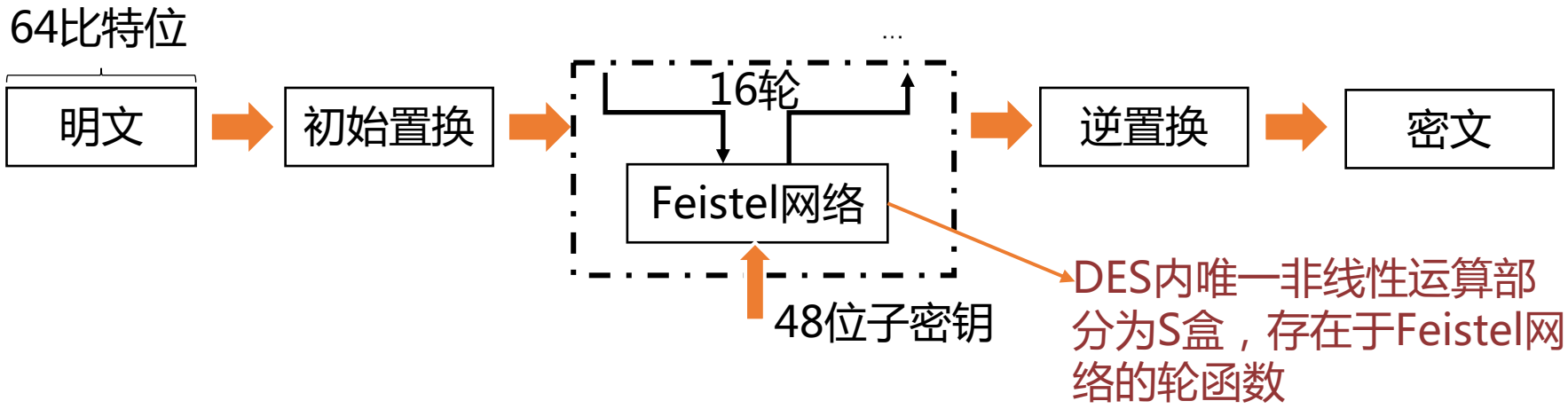
- 一类对称加密算法：将明文进行**分组**，将每个明文分组作为整体进行加解密

- Feistel密码结构：一种用于构造分组密码的密码结构
  - **扩散**：明文每一位影响密文许多位
  - **混淆**：隐藏密文与密钥统计关系
  - 加解密遵循相似运算流程



DES

- 分组长度64位；有效密钥长度56位（通过奇偶校验扩展为64位）
- 基于Feistel网络结构：



OpenSSL中的DES

- 头文件：**openssl/des.h**

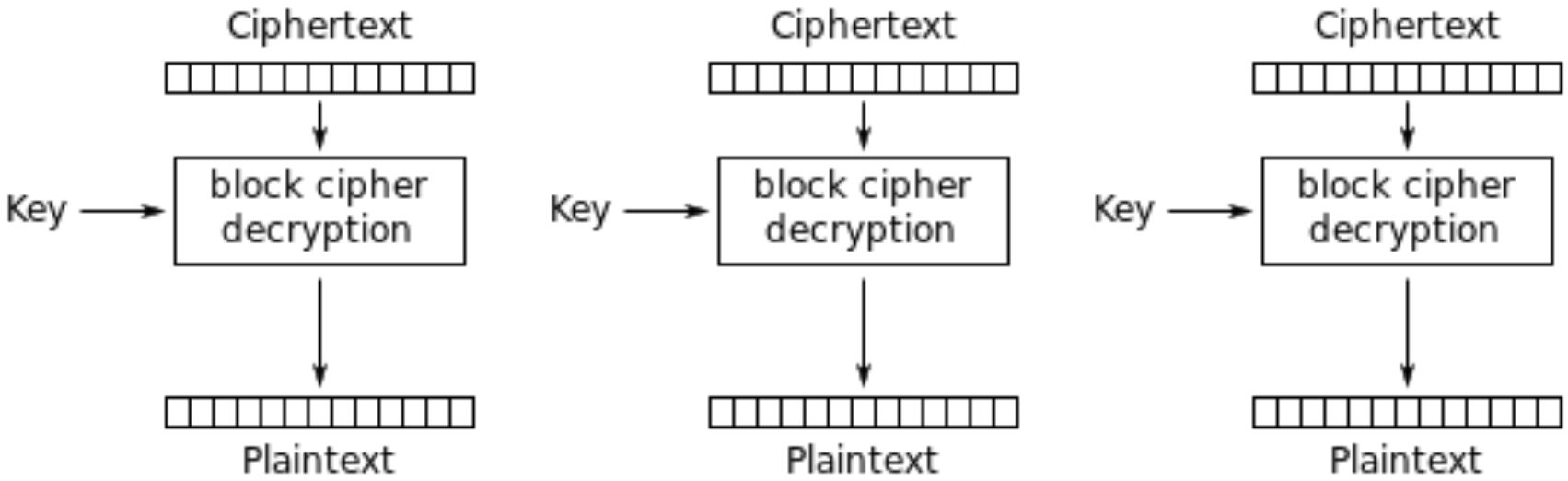
```
DES_cblock key; // DES_cblock结构: typedef unsigned char des_cblock[8];
DES_random_key(&key); // 随机产生密钥

DES_key_schedule keys;
DES_set_key_checked(&key, &keys); // 随机子密钥

DES_cblock plaintext, ciphertext;
DES_ecb_encrypt(&plaintext, &ciphertext, &keys, DES_ENCRYPT);
DES_ecb_encrypt(&plaintext, &ciphertext, &keys, DES_DECRYPT);
```

分组密码应用模式—ECB

➢使用分组密码处理**包含多个分组**长度的数据的加解密操作

➢ECB模式：

- **并行加密**
- **并行解密**
- **随机访问**



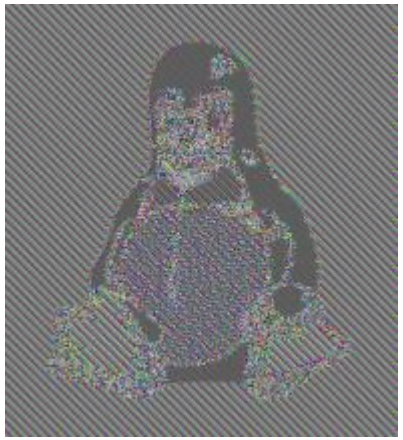
Electronic Codebook (ECB) mode decryption

分组密码应用模式—ECB

- 使用分组密码处理**包含多个分组**长度的数据的加解密操作

- ECB模式：
  - **并行加密**
  - **并行解密**
  - **随机访问**
  - **安全隐患**

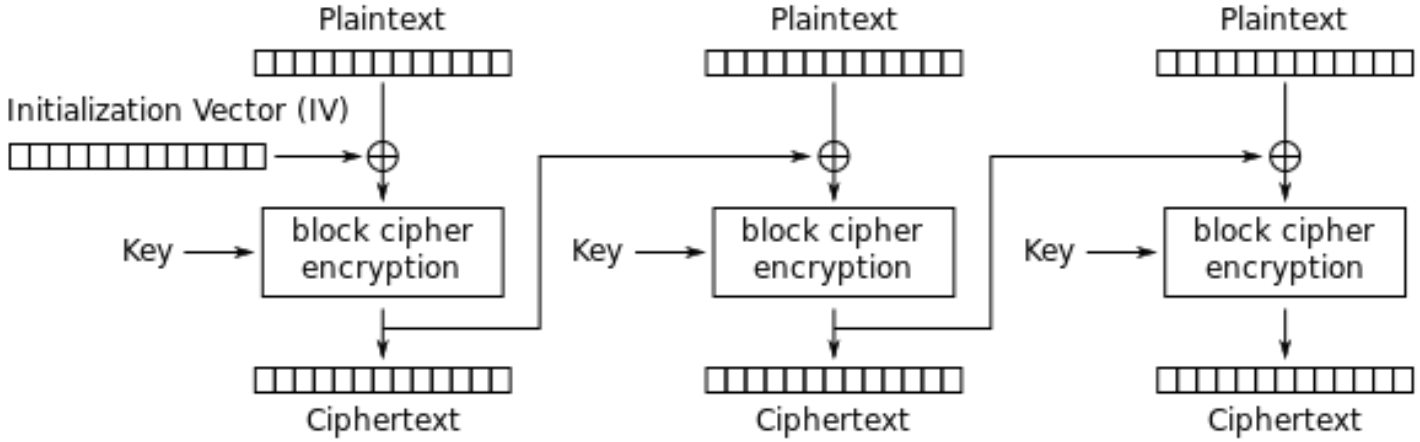
原始图像



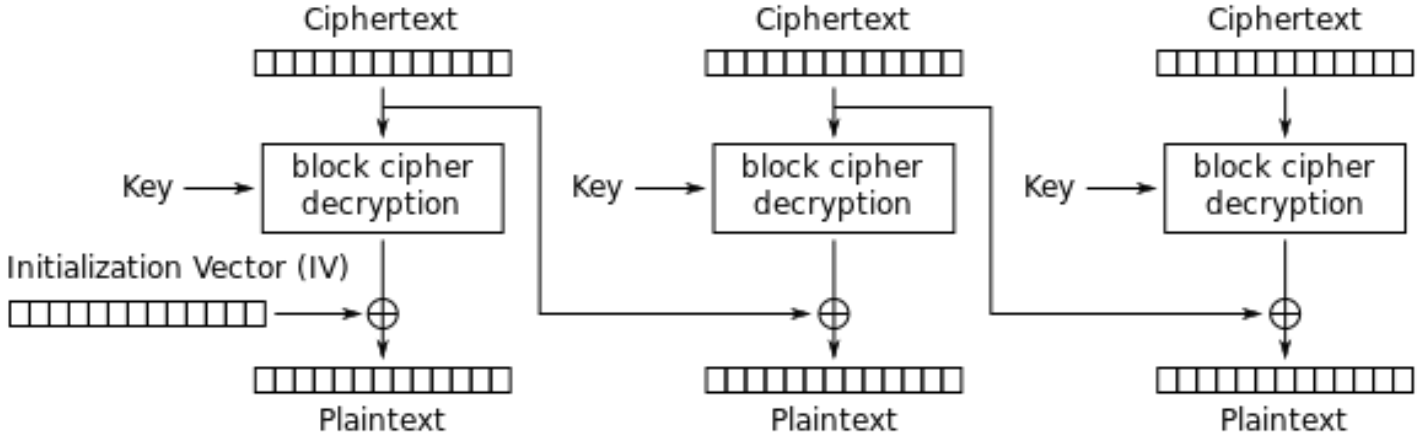
ECB加密结果

分组密码应用模式—CBC

- CBC模式：
  - 依赖**初始向量IV**
    - 公开，随机
  - **串行加密**
  - **并行解密**
  - **随机访问**



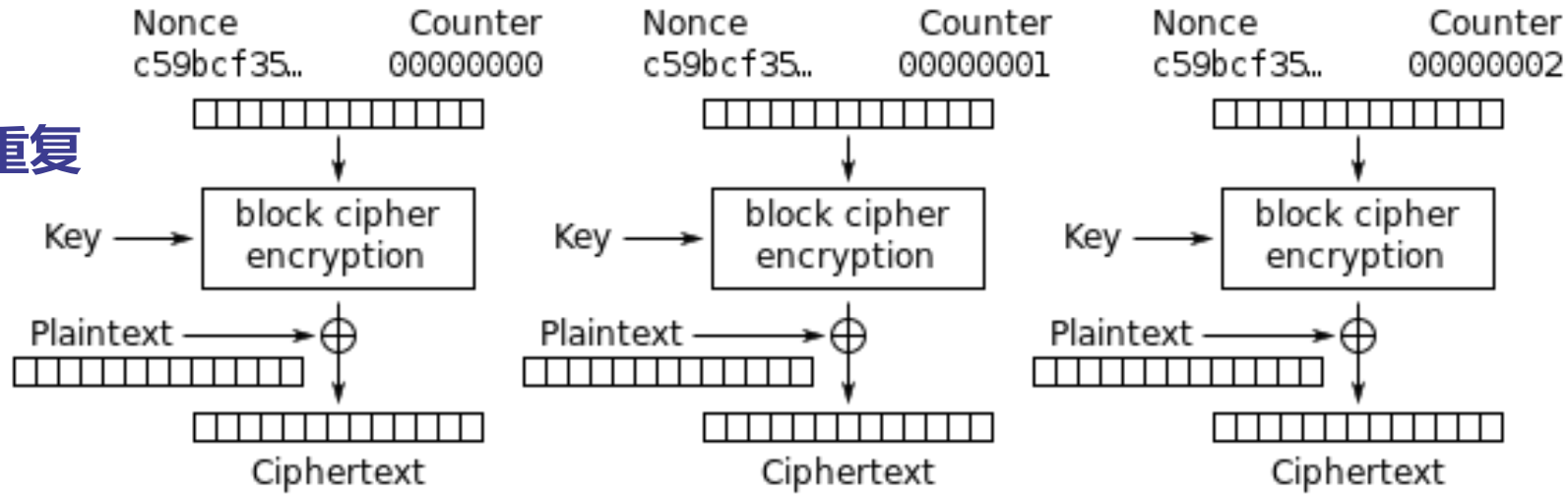
Cipher Block Chaining (CBC) mode encryption



Cipher Block Chaining (CBC) mode decryption

分组密码应用模式—CTR

- CTR模式：
  - 依赖**Nonce**
    - 公开，随机，**非重复**
  - **并行加密**
  - **并行解密**
  - **随机访问**
  - **预加密**[Speculative Encryption, FAST'19]



Counter (CTR) mode encryption

哈希函数：SHA1

哈希函数

- 将**任意长度**数据内容映射为固定长度哈希值
  - 特性：单向性、抗碰撞
  - 应用：数据完整性检测（防篡改）

	SHA1	SHA224	SHA256	SHA384	SHA512
消息摘要长度	160	224	256	384	512
消息长度	$< 2^{64}$	$< 2^{64}$	$< 2^{64}$	$< 2^{128}$	$< 2^{128}$
分组长度	512	512	512	1024	1024
字长度	32	32	32	64	64
步骤数	80	64	64	80	80

SHA1算法

- 预处理：填充、分块
- 分割字：将每个分块分割并扩展，共计产生80个字
  - 字长32比特位
- 定义5个寄存器，基于字内容进行变换，最终形成SHA1哈希
  - 参考：SHA1伪代码

OpenSSL中的SHA1

- 头文件：**openssl/sha.h**

```
/* Case：消息能够在内存中存放 */
/* 定义msg为消息，msg_len为消息长度，hash为至少20字节缓冲池 */
SHA(msg, msg_len, hash);

/* Case：消息难以在内存中存放 */
/* 定义chunk为消息的每个切块，chunk_len为chunk长度 */
SHA_CTX ctx;
SHA1_Update(&ctx, chunk, chunk_len);
SHA1_Final(hash, &ctx);
```

课程作业

RC4

- RC4加密&解密

```
// 该函数实现RC4加密算法功能
// 参数：data - 输入的明文字符串；secret_key - 密钥
// 返回值：string类型，返回加密结果。如果输入数据异常，则返回空字符串并退出
string rc4_encrypt(string data, string secret_key) {
    // TODO: 在此处实现你的代码
    return "TODO";
}

// 该函数实现RC4解密算法功能
// 参数：data - 输入的密文字符串；secret_key - 密钥
// 返回值：string类型，返回解密的结果。如果输入数据异常，则返回空字符串并退出
string rc4_decrypt(string data, string secret_key) {
    // TODO: 在此处实现你的代码
    return "TODO";
}
```

DES

- DES加密&解密：**使用convert_string_to_des_block函数**

```
// 该函数实现DES加密算法功能
// 参数：plain - 输入的明文字符串；secret_key - 密钥
// 返回值：string类型，返回加密的结果
string des_encrypt(string plain, string secret_key) {
    // TODO: 在此处实现你的代码
    return "TODO";
}

// 该函数实现DES解密算法功能
// 参数：cipher - 输入的密文字符串；secret_key - 密钥
// 返回值：string类型，返回解密的结果
string des_decrypt(string cipher, string secret_key) {
    // TODO: 在此处实现你的代码
    return "TODO";
}
```

SHA1

- SHA1哈希函数

```
// 该函数实现SHA1 hash算法功能
// 参数：msg - 输入的字符串
// 返回值：string类型，返回sha1消息摘要结果
string sha1_digest(string msg) {
    // TODO: 在此处实现你的代码
    return "TODO";
}
```