信息安全基础综合设计实验

基础知识

进制转换

2进制转化为10进制

```
unsigned int binary_to_decimal(std::string input){
    int n=input.length();
    unsigned int res=0;
    int i;
    for(i=n-1;i>=0;i--){
        if(input[i]!='0'&&input[i]!='1') return 0;
        res+=(input[i]-'0')*pow(2,n-i-1);
        if(res>=UINT_MAX) break;
    }
    return res;
}
```

10进制转化为2进制

```
std::string decimal_to_binary(unsigned int input){
    string s;
    if(input==0)s="0";
    while(input>0){
        if(input%2==0)s+='0';
        if(input%2==1)s+='1';
        input/=2;
    }
    reverse(s.begin(),s.end());
    return s;
}
```

模指数运算

```
// a: 输入底数
// e: 输入指数
// m: 输入模数
// 返回: a^e mod m
unsigned int mod_exp(unsigned int a, unsigned int e, unsigned int m){
    unsigned int res=1;
    while(e!=0){
        if(e%2==1){
            res=(res*a)%m;
        }
        a=((a%m)*(a%m))%m;
        e/=2;
    }
    return res;
}
```

素性测试

基于Eratosthenes筛选的素性测试

目标：产生最小N个素数（不适用于计算某个范围内全部素数）

方法：

（1）确定待筛选集合{2, 3,..., n^1/2}；

（2）基于{2, 3,..., n^1/2}的Eratosthenes筛选

①取第一个素数2, 划去{2,..., N}中除2以外所有2的倍数

②大于2的第一个正整数(即3)被认定为素数, 在余下的整数中划去除3以外所有3的倍数

③循环此过程直到找到{2, 3,..., N}中的所有素数

（3）筛选后元素与n整除判定

```
// a: 输入测试正整数
// 返回: true如果a为素数；false如果a不为素数
bool prime_test(unsigned int a){
    if(a==0||a==1)return false;
    unsigned int n=(unsigned int)sqrt(a);
    vector<int> set(n+1,1);
    int i,j;
    for(i=2;i<n;i++){
        if(set[i]==0) continue;
        j=i*i;
        while(j<=n){
            set[j]=0;
            j+=i;
        }
    }

    bool bo=true;
    for(i=2;i<=n;i++){
        if(set[i]==1){
            if(a%i==0){
                bo=false;
                break;
            }
        }
    }
    return bo;
}
```

Miller-Rabin素性测试

素数的两个性质：

➢性质 I：任意素数n可表示为**$n = 2^k q + 1$**，k >= 0，q为奇数

- 特殊情况：n = 2时，$2 = 2^0 * 1 + 1$

➢性质 II：n是素数，a是小于n的正整数，则**$a^2 \bmod n = 1$当且仅当$a \bmod n = 1$或$a \bmod n = n - 1$**

- 充分性：$a^2 \bmod n = (a \bmod n) * (a \bmod n) \bmod n = 1$
- 必要性：$a^2 \bmod n = 1$，则$a^2 - 1 \bmod n = 0$;
  即$(a+1)(a-1) \bmod n = 0$;
  由于n为素数，因此$a+1 \bmod n = 0$或$a-1 \bmod n = 0$;
  即$a \bmod n = n-1$或$a \bmod n = 1$

Miller-Rabin算法原理：

➢$n = 2^k q + 1$（k > 0，q为奇数）是大于2的**素数**，a是大于1且小于n-1的整数，如下**两个条件之一**成立

- $a^q \bmod n = 1$ **序列所有项均为1**
- 存在j（j >= 1且j <= k），满足$a^{2^{j-1}q} \bmod n = n-1$ **序列存在一项为n-1，使之后所有项均为1**

> ➢ 费马小定理：$a^{n-1} \bmod n = a^{2^k q} \bmod n = 1$
> ➢ 序列：$a^q \bmod n, a^{2q} \bmod n, \ldots, a^{2^{k-1}q} \bmod n, a^{2^k q} \bmod n$
> - **后一项恰为前一项的平方：**$a^{2^i q} \bmod n = [(a^{2^{i-1}q} \bmod n)^2] \bmod n$
> - **最后一项为1**

方法：

➢**Miller-Rabin**(n)

- 确定整数k和q，满足**$n = 2^k q + 1$**
- **随机**选择整数a，满足a > 1且a < n-1
- 如果**$a^q \bmod n = 1$**，返回**“不确定”**（可能是素数）
- 如果存在**$a^{2^{j-1}q} \bmod n = n-1$** (j = 1, 2,..., k)，返回**“不确定”**
- 返回“合数”

➢通过Miller-Rabin素性测试的数**不一定**是素数；无法通过Miller-Rabin素性测试的数一定不是素数（必要条件）

```
// 返回: "not_prime" - 表示一定不是素数
// "uncertain" - 表示不一定是素数，即无法确定
std::string miller_rabin_prime_test(unsigned int n, unsigned int a) {
    if(!(a>1&&a<n-1)){
        string error="error";
        return error;
    }
    string not_prime="not_prime";
```

```
    string uncertain="uncertain";

    //计算k,q
    unsigned int n_temp=n-1,k=0,q;
    while(n_temp){
        if(n_temp%2==0){
            n_temp/=2;
            k++;
        }else{
            q=n_temp;
            break;
        }
    }

    bool bo=true;
    int i;
    //判断 a^q mod n ?= 1, 即temp ?= 1
    unsigned int temp=1;
    for(i=0;i<q;i++){
        temp=(temp*a)%n;
    }
    if(temp==1||temp==n-1) bo=false;

    //判断 存在a^[(2^j-1)*q] mod n ?= n-1
    if(bo){
        for(i=2;i<=k;i++){
            temp=(temp*temp)%n;
            if(temp==n-1){
                bo=false;
                break;
            }
        }
    }
    if(bo) return not_prime;
    return uncertain;
}

// 每一轮随机选择a进行测试，存在某一轮无法通过，返回"not_prime"；
// repeat_times轮均能够通过测试，返回“uncertain”
std::string miller_rabin_multiple_test(unsigned int n, unsigned int
repeat_times) {
    string not_prime="not_prime";
    string uncertain="uncertain";
    srand((int)time(0));
    int i;
    for(i=0;i<repeat_times;i++){
        unsigned int a=rand()%(n-1);
        if(!(a>1&&a<n-1)){
            i--;
            continue;
        }
        if(miller_rabin_prime_test(n,a)=="not_prime"){
            return not_prime;
        }
    }
    return uncertain;
}
```

乘法逆元

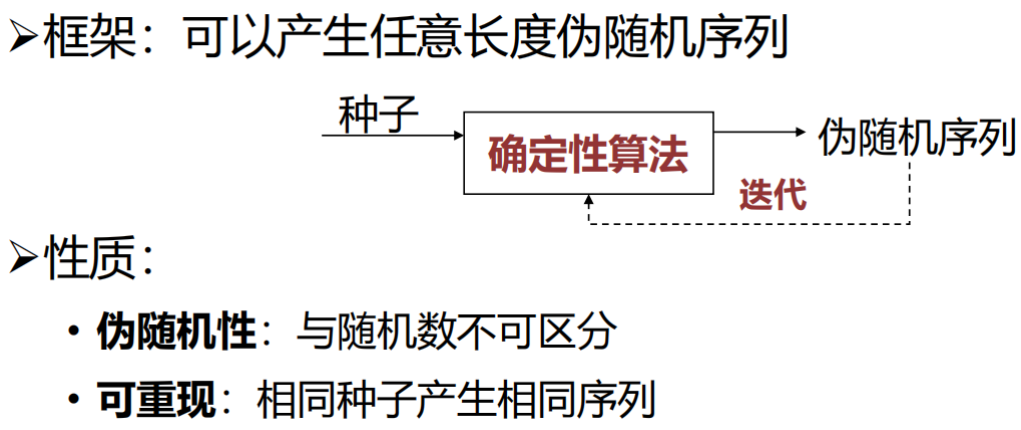
```
// 参数：
// a - 需要求逆元的数
// m - 模
// 返回值：std::int
// 返回值说明：返回a关于m乘法逆元；不存在返回-1
int ex_gcd(int a, int m, int &x, int &y) {
    if (m == 0) {
        x = 1; y = 0;
        return a; // 到达递归边界返回上一层
    }
    int r = ex_gcd(m, a % m, x, y);
    int t = x; x = y; y = t - a / m * y;
    return r; // 得到最大公约数
}

int euclid_mod_reverse(int a, int m) {
    if (a<=0||m<=0) return -1;
    int x,y,r;
    r=ex_gcd(a,m,x,y);
    if(r!=1) return -1;
    if(x<0) x+=m;
    return x;
}
```

伪随机数生成

随机数在密码算法和协议中广泛应用：认证协议、产生会话密钥、流密码

随机数具有随机性：均匀分布、难以重现

伪随机数通过伪随机数生成器产生，近似随机数



线性同余伪随机数生成

➢早期rand函数基于**线性同余**算法实现

➢迭代式：$X_{i+1} = aX_i + c \bmod m$

- 参数：a（乘数）、c（增量）、m（模数）
- 种子：X_0

安全性

➢评价标准

- **全周期**：{0, 1,..., m-1}中任意数都可能被生成
- **不可预测**：无法基于$X_0, X_1,..., X_{i-1}$推断X_i

➢对于任意参数配置，线性同余生成器无法满足全周期

- 例如：m = 32，a = 7，c = 0，产生序列{7, 17, 23, 1, 7,...}

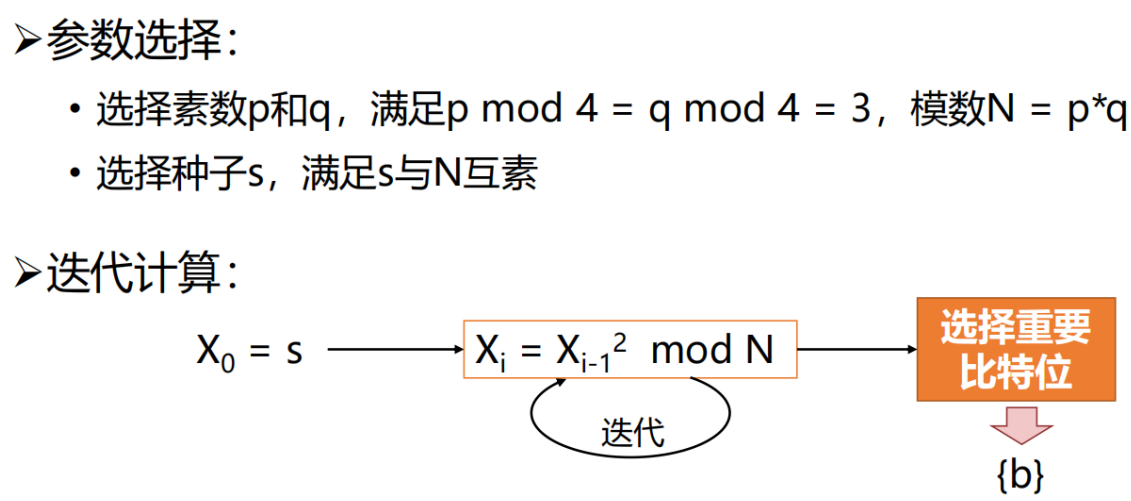
➢参数(a, c, m) + 伪随机数X_i → 后续伪随机数序列{$X_{i+1}, X_{i+2},...$}

- 增强方法：使用系统时钟修正增量

```
// 实现线性同余伪随机数生成器算法，采用固定参数配置：
// a = 1103515245; c = 12345; m = 2^31
// 返回线性同余算法产生的下一个unsigned int伪随机数
// 注：该函数运算将基于lcg_srand设置的种子；
// 如果未通过lcg_srand设置种子，默认种子为1
unsigned int gseed=1;
void lcg_srand(unsigned int seed){// 设置线性同余种子
    gseed=seed;
}

unsigned int lcg_rand(){
    unsigned int a=1103515245,c=12345,m=pow(2,31);
    unsigned int res=(a*gseed+c)%m;
    lcg_srand(res);
    return res;
}
```

BBS伪随机数生成

原理：



安全性：大数难分解问题

```
// 实现BBS伪随机数生成器算法，采用固定参数配置：
// p = 11; q = 19; s（种子） = 3
// 返回BBS算法产生的下一个unsigned int伪随机数
// 参数flag：标识选择重要比特位的方式
// flag = 0 - 选择最低比特位
// flag = 1 - 选择奇校验位
// flag = 2 - 选择偶校验位
// 提示：执行32轮迭代，将产生的32伪随机比特转换为伪随机数

//2进制转化为10进制
unsigned int binary_to_decimal(std::string input){
    int n=input.length();
    unsigned int res=0;
    int i;
    for(i=n-1;i>=0;i--){
        if(input[i]!='0'&&input[i]!='1') return 0;
        res+=(input[i]-'0')*pow(2,n-i-1);
        if(res>=UINT_MAX) break;
    }
    return res;
}

//10进制转化为2进制
std::string decimal_to_binary(unsigned int input){
    string s;
    if(input==0)s="0";
    while(input>0){
        if(input%2==0)s+='0';
        if(input%2==1)s+='1';
        input/=2;
    }
    reverse(s.begin(),s.end());
    return s;
}

unsigned int bbs_rand(int flag){
```

```
    unsigned int p=11,q=19,s=3;
    unsigned int N=p*q;
    int i,j;
    vector<unsigned int> x_temp;
    x_temp.push_back(s);
    for(int i=1;i<=32;i++){
        unsigned int temp;
        temp=(x_temp[i-1]*x_temp[i-1])%N;
        x_temp.push_back(temp);
    }
    vector<string> x_string;
    for(int i=1;i<=32;i++){
        x_string.push_back(decimal_to_binary(x_temp[i]));
    }

    string res;
    if(flag==0){
        for(i=0;i<32;i++){
            string temp=x_string[i];
            int n=temp.length();
            res=res+temp[n-1];
        }
    }else if(flag==1){
        for(i=0;i<32;i++){
            string temp=x_string[i];
            int n=temp.length();
            int num=0;
            for(j=0;j<n;j++){
                if(temp[j]=='1') num++;
            }
            if(num%2==0){
                res=res+'1';
            }else{
                res=res+'0';
            }
        }
    }else if(flag==2){
        for(i=0;i<32;i++){
            string temp=x_string[i];
            int n=temp.length();
            int num=0;
            for(j=0;j<n;j++){
                if(temp[j]=='1') num++;
            }
            if(num%2==0){
                res=res+'0';
            }else{
                res=res+'1';
            }
        }
    }else{
        return 0;
    }
    return binary_to_decimal(res);
}
```

时间比较

```
// 通过调用lcg_rand()和bbs_rand()，分别产生10个unsigned int
// 伪随机数，比较两种算法的运行时间（精确至微秒级），并在屏幕输出
void rand_time(){
    float time_use1=0,time_use2=0;
    struct timeval start;
    struct timeval end;

    //lcg
    gettimeofday(&start,NULL);
    int i;
    for(i=0;i<10;i++){
        unsigned int temp=lcg_rand();
    }
    gettimeofday(&end,NULL);
    time_use1=(end.tv_sec-start.tv_sec)*1000000+(end.tv_usec-start.tv_usec);
    cout<<"lcg_rand time:"<<time_use1<<"ms"<<endl;

    //bbs
    gettimeofday(&start,NULL);
    for(i=0;i<10;i++){
        unsigned int temp=bbs_rand(0);
    }
    gettimeofday(&end,NULL);
    time_use2=(end.tv_sec-start.tv_sec)*1000000+(end.tv_usec-start.tv_usec);
    cout<<"bbs_rand time:"<<time_use2<<"ms"<<endl;
}
```

OpenSSL

gcc/g++ <源文件> -o <可执行文件> -lcrypto

模指数运算

```
// 该函数用于进行大数模指数运算
// 参数：string类型，求解a^e mod m，表示为10进制数字符串
// 返回值：string类型，返回计算的结果，表示为10进制数字符串
string mod_exp(string a, string e, string m) {
    BIGNUM *A, *E, *M, *R;
    BN_CTX *ctx;
    ctx = BN_CTX_new();
    A = BN_new();
    E = BN_new();
    M = BN_new();
    R = BN_new();
    BN_dec2bn(&A, a.data());
    BN_dec2bn(&E, e.data());
    BN_dec2bn(&M, m.data());
    BN_mod_exp(R, A, E, M, ctx);
    string ret = BN_bn2dec(R);
    BN_free(A);
    BN_free(E);
    BN_free(M);
```

```
    BN_free(R);
    BN_CTX_free(ctx);
    return ret;
}
```

乘法逆元

```
// 该函数用于进行大数求乘法逆元
// 参数：string类型，求解a关于m的乘法逆元，表示为10进制数字符串
// 返回值：string类型，返回计算的结果，表示为10进制数字符串
string mod_inverse(string a, string m) {
    BIGNUM *A, *M, *R;
    BN_CTX *ctx;
    ctx = BN_CTX_new();
    A = BN_new();
    M = BN_new();
    R = BN_new();
    BN_dec2bn(&A, a.data());
    BN_dec2bn(&M, m.data());
    BN_mod_inverse(R, A, M, ctx);
    string ret = BN_bn2dec(R);
    BN_CTX_free(ctx);
    BN_free(A);
    BN_free(M);
    BN_free(R);
    return ret;
}
```

非对称密码

RSA（大数难分解困难问题）

步骤：

（1）选取大素数p、q，N=p*q，φ(N)=(p-1)(q-1)

（2）选取整数e，1<e<φ(N)，使得gcd(φ(N),e)=1

（3）计算d，使ed=1 mod φ(N)

加密：

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

签名：（完整性、不可伪造性、不可抵赖性）

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

RSA密钥载入

```
// 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
#define PUBLICKEY "../keys/public.pem"
#define PRIVATEKEY "../keys/private.pem"

RSA *rsa_private_key = NULL; // rsa私钥
RSA *rsa_public_key = NULL; // rsa公钥

bool load_RSA_keys() {
    FILE *fp = NULL; // 初始化文件指针
    if ((fp = fopen(PUBLICKEY, "r")) == NULL){
        return false;
    }
    rsa_public_key = PEM_read_RSA_PUBKEY(fp, NULL, NULL, NULL);
    if((fp = fopen(PRIVATEKEY, "r")) == NULL){
        return false;
    }
    rsa_private_key = PEM_read_RSAPrivateKey(fp, NULL, NULL, NULL);
    if(!rsa_public_key || !rsa_private_key){
        return false;
    }
    fclose(fp);
    return true;
}
```

RSA加解密

```
// 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
//加密
string RSA_Encryption(string plaintext)
{
    load_RSA_keys();
    char* cipher = (char*)malloc(RSA_size(rsa_public_key));
    memset(cipher, 0, RSA_size(rsa_public_key));
    int len_cipher = RSA_public_encrypt(plaintext.size(), (unsigned
char*)plaintext.c_str(),(unsigned char*)cipher, rsa_public_key,
RSA_PKCS1_PADDING);
    return string(cipher, len_cipher);
}

// 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
//解密
string RSA_Decryption(string ciphertext)
{
    load_RSA_keys();
    char* plain = (char*)malloc(RSA_size(rsa_private_key));
    memset(plain, 0, RSA_size(rsa_private_key));
    int len_plain = RSA_private_decrypt(ciphertext.size(), (unsigned
char*)ciphertext.c_str(),(unsigned char*)plain, rsa_private_key,
RSA_PKCS1_PADDING);
```

```
    return string(plain, len_plain);
}
```

RSA签名

```
// 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
//签名
string RSA_signature_signing(string input)
{
    load_RSA_keys();
    char* plain = (char*)malloc(RSA_size(rsa_private_key));
    memset(plain, 0, RSA_size(rsa_private_key));
    int len_plain = RSA_private_decrypt(input.size(), (unsigned
char*)input.c_str(),(unsigned char*)plain, rsa_private_key, RSA_PKCS1_PADDING);
    return string(plain, len_plain);
}

// 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
//验证
bool RSA_signature_verify(string message, string signature)
{
    load_RSA_keys();
    char* cipher = (char*)malloc(RSA_size(rsa_public_key));
    memset(cipher, 0, RSA_size(rsa_public_key));
    int len_cipher = RSA_public_encrypt(signature.size(), (unsigned
char*)signature.c_str(),(unsigned char*)cipher, rsa_public_key,
RSA_PKCS1_PADDING);
    string decryptVal((char*)cipher);
    if (message == decryptVal) return true;
    else return false;
}
```

对称密码

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

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

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

```
// 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：扩充状态数组，加密明文数据

```
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]];
}
```

```
// ================ RC4 Start ================================

// 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，返回加密的结果。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) {
    int secret_key_len=secret_key.length();
    int data_len=data.length();
    if(secret_key_len==0||data_len==0){
        return "";
    }

    RC4_KEY key;
    RC4_set_key(&key,secret_key_len,(unsigned char*)secret_key.c_str());

    unsigned char *outdata;
    outdata=(unsigned char*)malloc(sizeof(unsigned char)*(data_len+1));
    memset(outdata,0,data_len+1);
    RC4(&key, data_len, (unsigned char*)data.c_str(),outdata);
    string res;
    res.append(reinterpret_cast<const char*>(outdata));
    return res;
}

// 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) {
    int secret_key_len=secret_key.length();
    int data_len=data.length();
```

```
    if(secret_key_len==0||data_len==0){
        return "";
    }

    RC4_KEY key;
    RC4_set_key(&key,secret_key_len,(unsigned char*)secret_key.c_str());

    unsigned char *indata;
    indata=(unsigned char*)malloc(sizeof(unsigned char)*(data_len+1));
    memset(indata,0,data_len+1);
    RC4(&key, data_len, (unsigned char*)data.c_str(),indata);
    string res;
    res.append(reinterpret_cast<const char*>(indata));
    return res;
}

// ================ RC4    End ================================
```

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

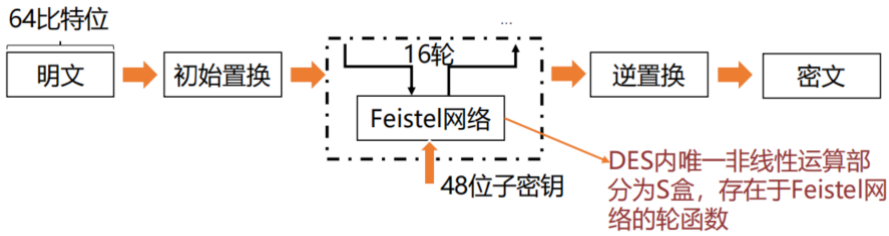
混淆：隐藏密文与密钥统计关系

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）

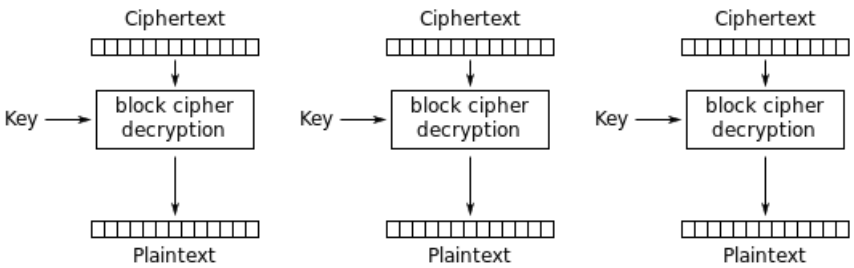
基于Feistel网络结构：



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

并行加密、并行解密、随机访问、（安全隐患）

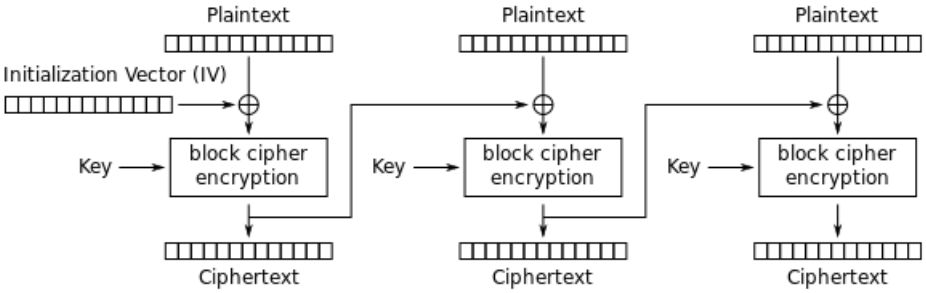


Electronic Codebook (ECB) mode decryption

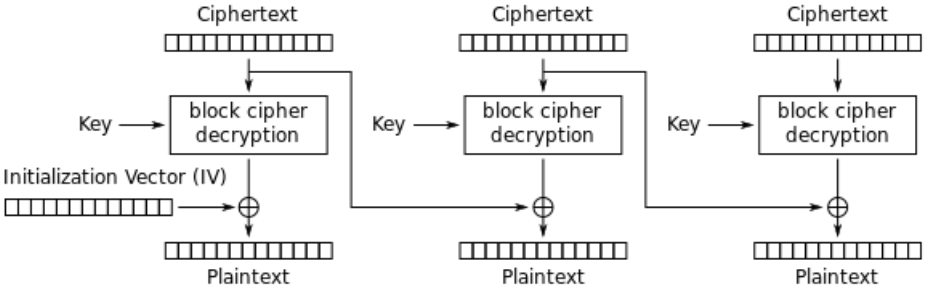
CBC

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

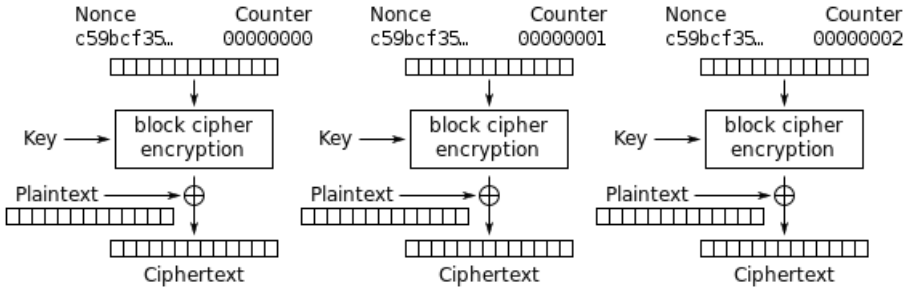


Cipher Block Chaining (CBC) mode decryption

CTR

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



Counter (CTR) mode encryption

```
// ================ DES Start ================================

// 请使用该函数将string类型转换为DES_cblock类型。（请勿更改此函数）
// - 该函数仅用于转换明文和密文内容，请不要把用此函数转换secret_key。
// - secret_key请使用`DES_string_to_key()`函数转换。
void convert_string_to_des_block(string str, DES_cblock &output);

// 该函数实现DES-ECB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
```

```
//  其他说明:
//      - 请使用`convert_string_to_des_block()`函数将string类型转换为DES_cblock类型。
//        若自行转换，可能导致测试无法通过
//      - secret_key使用前请用`DES_string_to_key()`设置key;
string des_encrypt(string plain, string secret_key) {
    DES_cblock key;
    DES_string_to_key(secret_key.c_str(), &key);
    DES_key_schedule schedule;
    DES_set_key_checked(&key,&schedule);

    DES_cblock input,output;
    convert_string_to_des_block(plain,input);

    DES_ecb_encrypt(&input, &output, &schedule, DES_ENCRYPT);
    return (char*)output;
}

// 该函数实现DES-ECB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
//  其他说明:
//      - 请使用`convert_string_to_des_block()`函数将string类型转换为DES_cblock类型。
//        若自行转换，可能导致测试无法通过
//      - secret_key使用前请用`DES_string_to_key()`设置key
string des_decrypt(string cipher, string secret_key) {
    DES_cblock key;
    DES_string_to_key(secret_key.c_str(), &key);
    DES_key_schedule schedule;
    DES_set_key_checked(&key,&schedule);

    DES_cblock output,input;
    convert_string_to_des_block(cipher,output);

    DES_ecb_encrypt(&output, &input, &schedule, DES_DECRYPT);
    return (char*)input;
}

// ================ DES   End ================================
void convert_string_to_des_block(string str, DES_cblock &output) {
    for(int i = 0; i < 8; ++i) {
        output[i] = str[i];
    }
}
```

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

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

```
// ================ SHA1 Start ================================

// 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) {
    char res[33]={'\0'};
    SHA1((const unsigned char *)msg.c_str(), msg.length(), (unsigned char*)res);
    return res;
}
// ================ SHA1   End ================================
```