密码学第二次大作业

题目1

2015011313 徐鉴劲 计54

Keccak算法的实现

Kaccak算法是SHA3的标准，它是一种采用了海绵结构（Sponge）的散列函数。海绵结构是说算法会将数据一轮一轮地放入到一个固定大小的数据池中，这样数据就会充分混合，然后再从中产生出固定长度的哈希结果。

不同采用海绵结构的算法，他们的区别之处就是在于每一轮吸收以后混合的函数不一样，Keccak的混合函数是这样定义的：

算法重复24次迭代，每一次包括五步（$\theta, \rho, \phi, \chi, Last$）：

θ: 将每一列混合起来，再进行移位混合。混合以后的结果反过来作用于海绵数据。

ρ和ϕ：移位和打乱。

χ：混合和写数据。

$Last$：异或一个数RC。

实现的算法在`keccak/SHA3`中。

1. `sha3_round`：每一轮keccak混合。
2. `keccak_f`：24轮`sha3_round`混合。
3. `sponge`：海绵结构，调用`padding`补足输入，再进行吸收，调用`keccak_f`和挤出。

通过这个网站上的SHA3进行了验证`https://asecuritysite.com/encryption/sha3`：

test和test_SHA3就是正确答案。

程序运行结果和预期相符，速度未经过优化，大约是30MB/s。

运行代码：

```
./main test <huge_file>
```

测试程序的速度需要使用一个大文件（可选参数），速度大约是30MB/s。

AES 128 CBC模式的实现

AES算法是分组算法，每次对一个固定分组的明文进行替换。

AES加密主要由下面几个操作构成：

- 密钥扩展
- 加轮密钥
- 字节替换
- 行移位
- 列混合

按照算法描述中的进行实现即可。

AES的解密是这些操作的反向。

算法并未进行其他的加速，经过测试能够达到26 MB/s的平均速度。

运行代码

编译：

```
mkdir build
cmake ../src -DCMAKE_BUILD_TYPE=Release
make
```

测试时间（需要放置一个huge_file在build目录下，并在此目录下执行）：

```
./test_aes
```

运行加密解密，0是加密，1是解密。

```
./main <key> <input> <output> <0/1>
```

题目3

找到一个最小的线性反馈移位寄存器，使之能够生成`0 0 1 0 1 0 1 0 0 1 0 0 0 1`

结果

通过运行`LSFRCrypto`中`build/main`可以得到解答。

级数为7，种子初始状态为：0111110，开关状态为：1101100。

用多项式表示为：$x^5 \oplus x^4 \oplus x^2 \oplus x \oplus 1$。

解题过程

线性反馈寄存器是每次产生一个新的最高位，然后将内部状态右移一位，丢弃的一位作为输出的一种器件。一个现行反馈寄存器可以由它的级数得到最高位的方法唯一确定。它有一个产生序列，根据不同的种子，可以截取这个产生序列的不同部分。

如果想要根据序列来反推LSFR的结构，采用穷举是一个方便的方法。采用搜索算法，用一个二进制数state表示初始状态（a_i），二进制数sw表示开关状态（c_i），枚举这两个二进制数的组合，然后然后模拟它的更新过程：

1. 更新最高位：$a_n = XOR\_PLUS(\mathbf{a}\&\mathbf{c})$。是a与c每一位与起来，然后将每一位的结果XOR起来。
2. 更新序列位：`res = (res << 1) + (state & 1)`，将state即将被替换出去的位放在输出里面。

整个算法的核心代码如下：

```
// 一共有14个序列长度
int M = 14;
int MASK =0b11111111111111;
int HIGH =0b10000000000000;
// 用二进制表示输出序列
int TAR = 0b00101010010001;
```

```
for(N=2; N<40; N++) {
    // 从两位开始枚举最小的级数
    printf("N=%d\n", N);
    for(sw = 0; sw < (1<<N); sw ++) {
        // 判断是否存在可能的种子，使之能够在这个开关的条件下生成目标序列。
        seed = judge_seq(sw);
        if(seed > -1){
            weight = sw;
            goto finish;
        }
    }
}
```

```
int judge_seq(int sw) {
    int state = 1;
    int ans = 0;
    int preres = 0;

    for(int it = 0; it < (1<<N); it ++) {
```

```
        // 枚举内部状态
        state = it;
        ans = 0;
        for(int i = 0; i < M; i ++) {
            // 求最高位
            int high = xor_plus(state & sw);
            // 求输出
            ans = (ans << 1) + (state & 1);
            // 判断是否需要早退出
            preres = (TAR >> (M-i-1));
            if(ans != preres) break;
            // 更新状态
            state = ((high << (N-1)) + (state >> 1));
        }
        // 成功找到
        if(ans == TAR) return it;
    }
    return -1;
}
```