六轮 DES 算法的差分分析

作业实现了四个部分

1. 6 轮 DES 算法的实现，主要用于产生合适的明密文对、最终破解时验证。
2. 差分分析表的生成
3. 根据给定密钥，随机生成明密文对，并检查是否符合选择明文攻击的条件。
4. 6 轮 DES 算法的差分分析，以及结果展示

6 轮 DES 算法的实现

const.h 文件中实现了各个置换表，以及对应的置换函数，包括：

ip：初始置换表

ext：F 函数中的扩展变换表。

sbox：F 函数中的 S 盒变换表。

perm：F 函数中的置换变换表。

pi：末置换表。

pc1：密钥生成的初始变换。

pc2：密钥生成过程中对每个子密钥的变换。

ull calcBase(const int *table, int input_size, int output_size, const ull &num);

置换函数

table: 置换表

input_size：输入的二进制数的位数

output_size：输出的二进制数的位数

num：输入的二进制数

输出：置换的结果

ull calcInvBase(const int *table, int input_size, int output_size, const ull &num)

置换函数的逆过程，参数同上。

des.h 文件中主要写了 DES 算法的过程。

void createSubKey(const ull &key, ull *sub_key)

子密钥产生算法

key：输入密钥

sub_key：输出的密钥数组

ull des(const ull &txt, const ull &key, bool encrypt)

DES 算法过程

txt：明文或者密文，取决于 encrypt 参数，输入为 64 位无符号整数

key：密钥

encrypt=0 表示解密，encrypt=1 表示加密
输出：encrypt=0 输出解密的明文，encrypt=1 输出加密的密文

差分分析表生成

did_table.h 文件中实现了差分分析表的生成。通过穷举，可以得到每个 S 盒的输入差分、输出差分所对应的 S 盒输入值。

class DidTable
void get(int index, int i, int j, vector<int> &ret)
对于第 index 个 S 盒，输入差分 i, 输出差分 j，返回查询的结果到数组 ret 中。
index：S 盒的编号
i：输入差分
j：输出差分
ret：返回值，是个数组，即所有可能的 S 盒输入值。

随机生成明密文对

《分组密码的设计与分析》一书中给出了两种三轮差分特征，其中一种如图 1 所示。

$$\Delta L_0 = 40080000 \quad \Delta R_0 = 04000000$$
$$\Delta L_1 = 04000000 \quad \Delta R_1 = 00000000 \quad p = \frac{1}{4}$$
$$\Delta L_2 = 00000000 \quad \Delta R_2 = 04000000 \quad p = 1$$
$$\Delta L_3 = 04000000 \quad \Delta R_3 = 40080000 \quad p = \frac{1}{4}$$

图 1：差分特征 1

该差分特征可以以 1/16 的概率破解到 J2、J5、J6、J7、J8 的正确比特密钥。
另外一种三轮差分特征如图 2 所示。

$$\Delta L_0 = 00200008 \quad \Delta R_0 = 00000400$$
$$\Delta L_1 = 00000400 \quad \Delta R_1 = 00000000 \quad p = \frac{1}{4}$$
$$\Delta L_2 = 00000000 \quad \Delta R_2 = 00000400 \quad p = 1$$
$$\Delta L_3 = 00000400 \quad \Delta R_3 = 00200008 \quad p = \frac{1}{4}$$

图 2： 差分特征 2

该差分特征以 1/16 的概率破解到 J1、J4 的正确比特密钥
分别构造两对明密文对（一共四个明密文对），分别满足上述特征之一，就可以破解 42 比特

的密钥（为了方便，程序中检测了第三轮加密的结果是满足ΔL_3和ΔR_3是否满足差分的条件，并不断随机直到筛选出了合适的明密文对），剩余的 14 位密钥需要穷举搜索实现。

在 hack.h 文件：
void constructCipherPlaintexts(const ull &key, int mode,
ull &inA, ull &inB, ull &outA, ull &outB);
随机构造一对明密文对，其中 mode=1 表示使用上述第一种差分特征构造，mode=2 表示使用上述第二种差分特征构造。

Key：密钥

Mode：提供两种差分特征以供选择

inA，inB，outA，outB：用于返回一对输入输出密钥。

差分分析过程

首先使用 constructCipherPlaintexts 函数构造出四个明密文对，每个明密文均为 64 位无符号长整数（unsigned long long 类型），其中两个符合图 1 中的差分特征，另两个符合图 2 中的差分特征。DesHacker 类实现了具体的差分分析过程，如图 3 所示。（参考《分组密码的设计与分析》）

输入：L_0R_0、$L_0^*R_0^*$、L_6R_6 和 $L_6^*R_6^*$，其中 $\Delta L_0=40080000$ 和 $\Delta R_0=04000000$。
(1) 计算 $C'=P^{-1}(R_6'\oplus 40080000)$。
(2) 计算 $E=E(L_6)$和 $E^*=E(L_6^*)$。
(3) 对 $j\in\{2,5,6,7,8\}$，计算 $\text{test}_j(E_j,E_j^*,C_j')$。

图 3：差分分析过程

根据分别满足图 1、图 2 差分特征的两对明密文对，就有 1/16 * 1/16 的概率破解 42 比特的密钥（为了方便，在构造过程中筛选掉了不能破解的明密文对，因此程序中的破解概率为 1），剩余的 14 位密钥需要穷举搜索实现。

class DesHacker
bool addCipherPlaintexts(ull in1, ull in2, ull out1, ull out2)
向 DesHacker 类中添加构造的明密文对，并验证是否符合差分特征。

In1，in2，out1，out2：构造的明密文对

返回为是否符合差分特征。

bool hack()
差分分析过程。返回为布尔变量，表示是否成功分析出密钥。

ull getKey()
返回差分分析得到的密钥。（在 hack 函数运行完毕后，执行 getKey 得到最终的密钥）

运算结果演示

给定密钥 0xF0F0F0F0F0F0F0F0，如图 4 为执行结果。

```
process: 109800/110592
process: 109900/110592
process: 110000/110592
process: 110100/110592
process: 110200/110592
process: 110300/110592
Success
key: F0F0F0F0F0F0F0F0

Process finished with exit code 0
```

图 4：差分分析结果

由于子密钥生成过程中 PC1 将 64 位密钥转换位了 56 位密钥，因此会有 8 个二进制位的损失。而这 8 位并不影响加密解密过程，因此破解来的密钥可能与给定的 64 位不同，但是仍然是正确的结果。