Web 安全技术第一次作业实验报告

陈铭涛

16340024

实验要求

完成一个DES算法的程序设计和实现，包括

✧算法原理概述；总体结构；模块分解；数据结构；源代码；编译运行结果

DES 算法原理概述

DES 是一种块加密算法，使用 64 位密钥（实际有效长度为 56 位），以 64 位为分组长度，输出每组密文长度同样为 64 位。DES 加密的主要过程为：

$$\mathrm{C} = \mathrm{E_k}(M) = IP^{-1} \cdot \mathrm{W} \cdot \mathrm{T_{16}} \cdot T_{15} \cdot \ldots \cdot T_1 \cdot IP(M)$$

其中，M 为 64 位明文块，$\mathrm{E_k}$为以 k 为密钥的加密函数，IP为 64 位初始置换，$\mathrm{T_1}, T_2, \ldots, T_{16}$为一系列的迭代变换，$\mathrm{IP}^{-1}$为 IP 的逆置换，最终的输出 C 为64位密文块。

对应的解密过程为：

$$\mathrm{M} = \mathrm{D_k}(C) = IP^{-1} \cdot W \cdot T_1 \cdot T_2 \cdot \ldots \cdot T_{16} \cdot IP(C)$$

其中$\mathrm{D_k}$为以 k 为密钥的解密函数。

$\mathrm{T_1}, T_2, \ldots, T_{16}$迭代变换的过程是首先将数据分为左右各 32 位的两个半块，每一轮迭代变换通过 Feistel 轮函数使用由密钥生成的 16 个 48 位子密钥 d 对右半块进行转换后与左半块进行异或作为新的右半块，每轮变换后将会交换左右半块的顺序。16 轮迭代后将左右半块进行交换。

每一轮的 Feistel 轮函数中，将输入的右半块数据进行扩展后与当前轮的子密钥进行异或获得 48 位数据。然后对这 48 位数据分成 8 组 6 位数据，每个分组使用 8 个不同的 S 盒进行查表转换，获得 8 个 4 位数据，连接并置换后即为函数结果。

S 盒的查表方法为使用 6 位数据中的第一和第六个比特组成行号，中间的四个比特组成列号，对应的 S 盒表中的数据就为查到的 4 位数据结果。

Feistel 结构使得 DES 的加密与解密的过程最主要的区别是 16 轮置换时使用的子密钥顺序是正序还是逆序，这特点使得加密与解密所实现的过程相似。

程序代码结构

提交的 DES 算法程序位于 src 文件夹，使用 Go 语言编写，包含了 4 个代码文件：

- main.go 为程序的主函数，对程序接收的参数进行分析并从标准输入中接收加

密解密的密钥，然后根据接收到的信息构造 DesCipher 对象并调用对应方法进行加密或解密。

- cipher/des.go 为 DES 加密的主要过程，提供通过密钥构造 DesCipher 对象的函数以及 DesCipher 的加密与解密的方法。
- cipher/staticData.go 为 DES 加密中所用的表数据，如 IP 置换表和 S 盒。
- cipher/des_test.go 为 DES 加密的测试，使用两个数据和两个密钥进行 ECB 模式和 CBC 模式的加密和解密效果进行测试，在 src/cipher 文件夹下执行 go test 命令进行。

程序实现概述

程序的 DES 算法的主要内容位于 cipher/des.go 文件下，实现的结构和函数具体如下：

```
type DesCipher struct {
    key     [8]byte
    subKey  [16][6]byte
    cbcMode bool
}
```

此为 DesCipher 的定义，数据成员包含其密钥，子密钥及加密模式（ECB 或 CBC）

```
func NewDesCipher(key []byte, cbcMode bool) (*DesCipher, error)
```

该函数用于构造一个 DesCipher 对象，接收一个 64 位的字节数组作为密钥和一个代表模式的布尔型变量，若需使用 CBC 模式则为 true，使用 ECB 模式则为 false。
返回两个变量为构造的 DesCipher 对象的指针，若接收的密钥长度不正确则返回为 nil 并返回一个 error 传递错误信息。

```
func (cipher *DesCipher) generateSubKey()
```

该方法跟据 DesCipher 对象的密钥进行 16 个 48 位子密钥地生成，生成的子密钥保存在对象的 subKey 变量中。

```
func (cipher *DesCipher) EncryptData(originData []byte) ([]byte, error)
func (cipher *DesCipher) DecryptData(originData []byte) ([]byte, error)
```

这两个 DesCipher 的方法用于加密与解密数据，接收需要进行加密或解密的数据作为参数，返回处理完毕后的数据，若处理过程中出错，则会返回错误信息。
因为 DES 算法的加密和解密过程较为相似，因此两个方法的实现都是直接调用下面的 processData 方法。

```
func (cipher *DesCipher) processData(originData []byte, encrypt bool) ([]byte, error)
```

该方法接收原始数据和代表加密还是解密的 bool 型变量作为参数，返回处理后的数据和错误信息。若此时为 cbc 模式，则将使用密钥作为初始向量进行处理。
该方法为 DES 加密算法的主体过程，实现中会调用下方的其他函数

```
func permute(origin, table []byte) []byte
```
该函数用于数据的置换，传入原始数据和置换表，返回置换后的数据。

```
func initialPermute(block []byte) []byte
```
该函数调用 permute 函数进行块的初始置换。

```
func feistel(right []byte, key [6]byte) []byte
```
此函数为迭代变换过程中使用的 feistel 函数，传入每一轮迭代时的右半块和 48 位子密钥作为参数，返回处理后的结果。

```
func getBit(arr []byte, index uint) byte
```
该函数接收一个字节切片和索引，返回该切片中索引位置的比特，返回形式为返回的字节的第一个比特。

```
func leftRotate(origin []byte, offset, firstOffset, lastLength uint) []byte
```
该函数将输入的字节数组循环左移数位，接收参数中 offset 为左移的位数，因为传入的数据的比特长度并不一定为 8 的倍数，因此需要指定 firstOffset 作为起始的比特在第一个字节中的位置已经 lastLength 作为最后一个字节中有效比特的数量。

```
func pkcs7Padding(blockData *[]byte) error
```
此函数使用 pkcs#7 进行字节填充，对于块中缺少的字节数，原地填充至满足块长度要求，填充的内容为缺失的字节个数，当输入的切片长度大于块长度时将返回错误。
程序数据加密过程中对获得数据的最后一个块进行此操作，若最后一块长度为 8 字节，则将再添加一块数据全为 8 的 8 字节块。

```
func pkcs7Unpadding(blockData *[]byte) error
```
此函数原地去除填充的字节，从后往前检查切片若发现存在要去除的字节的值不为要去除的字节数量时，不会进行字节去除的操作。

程序的编译与运行

提交的压缩包中 bin 文件夹已经包含了程序在 MacOS(Darwin), Windows 以及 Linux 下的可执行文件，若有需要也可根据下面的方法进行编译。

程序的编译方式为在解压后的 src 文件夹中执行命令 go build 即可在该文件夹中编译获得可执行文件 des（Windows 下为 des.exe）。

主程序的使用方式如下：

```
./des {encrypt|decrypt} [-p|--progress] [-s|--silent] [-cbc(default ecb mode)] [{-i|--input} filename]
```

各参数意义如下：

1. `{encrypt|decrypt}`为必须指定的选项，用于表明程序为加密还是解密模式，可使用 e 或 d 作为简写。

2. `[-p|--progress]`为可选的选项，若指定该选项则程序处理时将根据当前处理包的数量按 1 秒的频率更新显示当前加密或解密进度。
3. `[-s|--silent]`为可选选项，程序默认在加密后显示加密完成的字节值内容，解密完成后以字符串形式显示解密获得的数据，若指定该选项则程序不会显示这些内容。
4. `[-cbc(default ecb mode)]`为可选选项，程序默认使用 ECB 模式进行数据的处理，使用该选项后程序将使用 CBC 模式进行处理，初始向量为输入的密钥。
5. `[{-i|--input} filename]`在-i 后接文件名作为加密或解密的数据输入。该选项在加密模式下为可选选项，若不指定程序将从标准输入中读取需要加密的内容，直至用户输入 EOF。在解密模式下则为必须指定的内容。

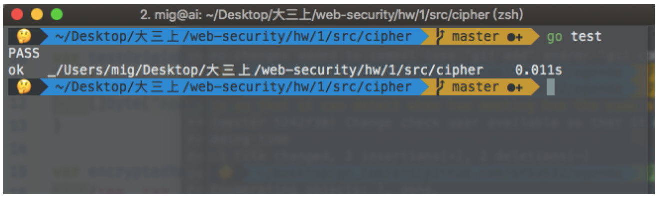
程序启动后将要求用户输入加密或解密的密钥，若密钥长度符合要求则将开始加密或解密的操作，加密完成后的结果将会写入文件 EncryptedData，解密完成后的数据将会写入文件 DecryptedData

程序测试效果

在 src/cipher 文件夹下执行命令 go test 可以运行一个简单的测试，测试将会对以下两个字符串使用密钥 12345678 和 abcdefgh 进行 ECB 模式下和 CBC 模式下加密与解密的测试。

```
"This is a test to see if this DES encryption implementation works."
"aaaaaaaáâäaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa"
```

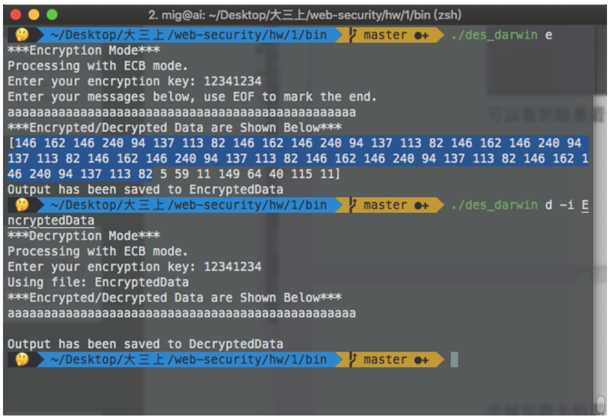
执行效果如下：



以下使用其他数据对程序进行加解密的测试：

1. 使用同样的密文块测试 ECB 和 CBC 模式的区别：

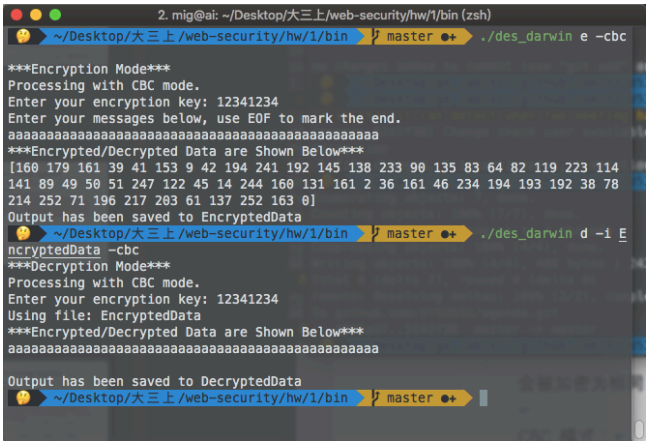
   ECB 模式：



   可以看到除最后一个块以外其他的密文块是重复的，表明了 ECB 模式下相同的明文块

会被加密为相同的密文块。

CBC 模式：



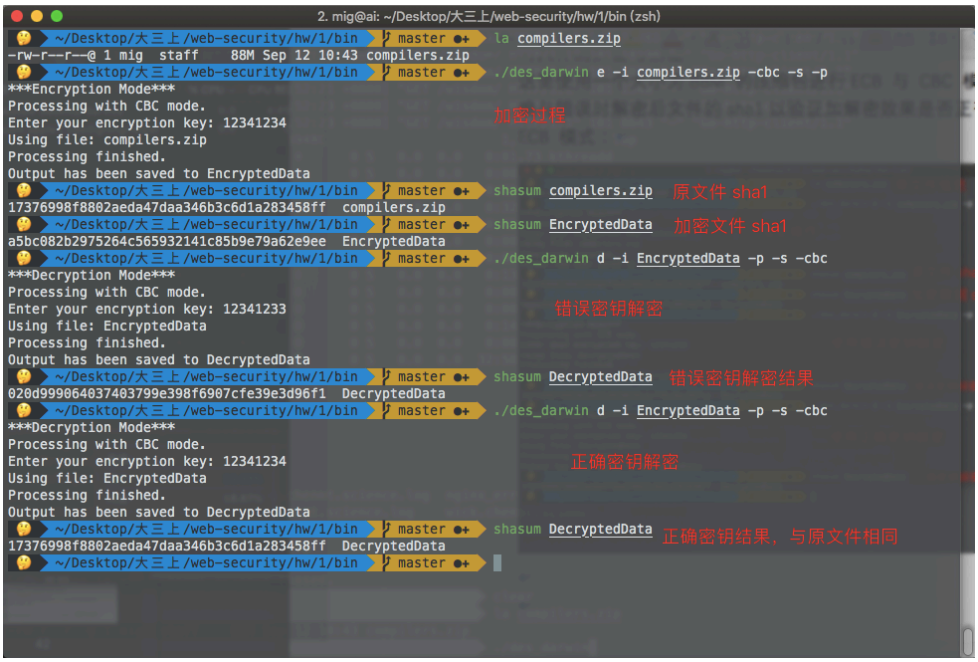
可以看到 CBC 模式下密文与明文块不再一一对应，因此破解更加困难，但是缺点是实时性不高。

2. 对文件的加密与解密

这里使用一个大小为 88M 的压缩包作为输入进行 ECB 与 CBC 模式下的测试，对比解密密钥正确与错误时解密后文件的 sha1 以验证加解密效果是否正确。

ECB 模式：

CBC 模式：



由此测试可验证程序对文件的加解密正确性。