《密码学基础原理》实验报告

课程：密码学基础原理　　实验名称：　AES 基本变换

姓名：　陈钦　　实验日期：　2022/10/25

学号：　2021131094　　实验报告日期：　2022/10/25

班级：　区块链工程 213 _

教师评语： 签名： 日期：	成绩：

一、实验名称

ＡＥＳ密钥基本变换

二、实验环境（详细说明运行的系统、平台及代码等）

1. 系统：go version go1.19 windows/amd64
2. IDE ：GoLand 2022.2.2

三、实验目的

（1）加深对 *AES* 算法的理解；

（2）阅读标准和文献，提高自学能力；

（3）加深对模块化设计的理解，提高编程实践能力。

四、实验内容、步骤及结果

１．实验内容

根据 AES 算法的原理，编写程序进行计算

２．实验步骤

（１）字节代替：输入一个数组，数组中的每一个数值的前一个字符和后一个字

符分别表示 S 盒的行和列。然后进行索引替换，返回输出数组

```go
// 字节代替
func subBytes_94(arr [16]byte) (newArr [16]byte) {
    for index, value := range arr {
        low := value & 0x0F          //取低4位
        high := (value >> 4) & 0x0f //取高4位
        newArr[index] = Sbox[16*high+low]
    }
    return newArr
}
```

（2）行移位：根据 AES 的行移位规则进行每一行移位。因为我是一维数组而部署二维数组，进行 for 循环替换则麻烦，于是我直接进行简单的替换

```go
func ShiftRows_94(arr [16]byte) (newArr [16]byte) {
    for index, _ := range arr {
        newArr[index] = arr[index]
    }
    //第一行不移位
    //第二行移位
    newArr[4] = arr[5]
    newArr[5] = arr[6]
    newArr[6] = arr[7]
    newArr[7] = arr[4]
    //第三行移位
    newArr[8] = arr[10]
    newArr[9] = arr[11]
    newArr[10] = arr[8]
    newArr[11] = arr[9]
    //第四行移位
    newArr[12] = arr[15]
    newArr[13] = arr[12]
    newArr[14] = arr[13]
    newArr[15] = arr[14]
    return newArr
```

（3）列混合：根据书本上的计算规则原理来进行编程。

columnMixCount 的三个方法：与 { 0 1 }，{ 0 2 }，{ 0 3 } 进行相乘

columnMixResult 的四个方法：固定矩阵的 4 行分别和输入矩阵的的列相乘情况

columnMix_94：首先将输入的数组进行重排，原因：列混合是在列方向进行，而输入的数组是行方向输入的。然后进行矩阵的相乘计算。因为用循环等方法进行编程不好理解，因此矩阵相乘我采用的是一个个列举出来。

```
84    // 列混合
85    func columnMix_94(arr [16]byte) [16]byte {...} //列混合主函数
127
128   func columnMixCount01_94(num byte) byte {...}
131
132   func columnMixCount02_94(num byte) byte {...}
139
140   func columnMixCount03_94(num byte) byte {...}
143
144   func columnMixResult01_94(num01 byte, num02 byte, num03 byte, num04 byte) byte {...}
148
149   func columnMixResult02_94(num01 byte, num02 byte, num03 byte, num04 byte) byte {...}
153
154   func columnMixResult03_94(num01 byte, num02 byte, num03 byte, num04 byte) byte {...}
158
159   func columnMixResult04_94(num01 byte, num02 byte, num03 byte, num04 byte) byte {...}
```

（4）轮密钥加：输入数组和子密钥对应的每一个数进行异或运行，返回新数组

```
164   // 轮密钥加
165   func AddRoundKey_94(arr [16]byte) (newArr [16]byte) {
166       //定义子密钥.
167       var arrKey [16]byte = [16]byte{0xa0, 0x88, 0x23, 0x2a, 0xfa, 0x54, 0xa3, 0x6c, 0xfe, 0x2c, 0x39,
168           0x76, 0x17, 0xb1, 0x39, 0x05}
169       for i := 0; i < 16; i++ {
170           newArr[i] = arrKey[i] ^ arr[i]
171       }
172       return newArr
173   }
```

3.实验结果

首先检验算法是否正确。使用书本上的例题。

根据图片显示，我所编写的程序是正确的。

```
字节代替：
 E6 B1 CA B7
 7F 1B 5B 12
 7C 7B 50 FD
 79  4 23 18
行移位 ：
 E6 B1 CA B7
 1B 5B 12 7F
 50 FD 7C 7B
 18 79  4 23
列混合 ：
 B2 10 C1 AC
 38 62 6E E7
 75 80 2C 5B
 4A 9C 23 80
轮密钥加：
 3A 9D 3F 52
  C 37 59 85
 3D F1 58 26
 B0 30 BE 2F
```

本实验的三个输入的输出为：

```go
func main() {
    //定义我们的输入
    fmt.Println( a...: "(1)当输入为：32 43 f6 a8 88 5a 30 8d 31 31 98 a2 e0 37 07 34:")
    var arr01 = [16]byte{0x32, 0x43, 0xf6, 0xa8, 0x88, 0x5a, 0x30, 0x8d, 0x31, 0x31,
        0x98, 0xa2, 0xe0, 0x37, 0x07, 0x34}
    AES_94(arr01)
    fmt.Println()
    fmt.Println()

    fmt.Println( a...: "(2)当输入为：19 3d e3 be a0 f4 e2 2b 9a c6 8d 2a e9 f8 48 08:")
    var arr02 = [16]byte{0x19, 0x3d, 0xe3, 0xbe, 0xa0, 0xf4, 0xe2, 0x2b, 0x9a, 0xc6, 0x8d, 0x2a, 0xe9, 0xf8, 0x48, 0
    AES_94(arr02)
    fmt.Println()
    fmt.Println()

    fmt.Println( a...: "(3)当输入为：20 21 13 10 94 00 00 00 00 00 00 00 00 00 00 0B:")
    var arr03 = [16]byte{0x20, 0x21, 0x13, 0x10, 0x94, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0
    AES_94(arr03)
```

```
(1)当输入为：32 43 f6 a8 88 5a 30 8d 31 31 98 a2 e0 37 07 34:
字节代替：
 23 1A 42 C2
 C4 BE  4 5D
 C7 C7 46 3A
 E1 9A C5 18
行移位 ：
 23 1A 42 C2
 BE  4 5D C4
 46 3A C7 C7
 18 E1 9A C5
列混合 ：
 C1 E3 3E CA
 96 BD 30 C6
 39 52 3F C7
 AD C9 73 CF
轮密钥加：
 61 6B 1D E0
 6C E9 93 AA
 C7 7E  6 B1
 BA 78 4A CA
```

```
(2)当输入为：19 3d e3 be a0 f4 e2 2b 9a c6 8d 2a e9 f8 48 08:
字节代替：
 D4 27 11 AE
 E0 BF 98 F1
 B8 B4 5D E5
 1E 41 52 30
行移位 ：
 D4 27 11 AE
 BF 98 F1 E0
 5D E5 B8 B4
 30 1E 41 52
列混合 ：
  4  6 D3 9A
 66 26 7A E0
 81 4C 48 CB
 E5 28 F8 19
轮密钥加：
 A4 8E F0 B0
 9C 72 D9 8C
 7F 60 71 BD
 F2 99 C1 1C
```

```
(3)当输入为：20 21 13 10 94 00 00 00 00 00 00 00 00 00 00 0B:
字节代替：
 B7 FD 7D CA
 22 63 63 63
 63 63 63 63
 63 63 63 2B
行移位 ：
 B7 FD 7D CA
 63 63 63 22
 63 63 63 63
 2B 63 63 63
列混合 ：
 98 44 5F E9
 FF FD 7D 48
 6F FD 7D 8B
 94 DA 41 C2
轮密钥加：
 38 CC 7C C3
  5 A9 DE 24
 91 D1 44 FD
 83 6B 78 C7
```

五、实验中的问题及心得

本人使用 go 语言进行编程，在字节代替的那一步参考了实验参考代码，其余步骤均根据个人对 AES 原理进行编写。因为有些步骤使用循环等方式编程并不利于理解，因此我采用了列举的方法进行编程，思路简单但代码繁多。

在编程过程中，主要思路是将一 AES 算法分为 4 个部分进行编程，使用 byte 数组接收每一个数，因为一个 byte 正好是 8 位，十分符合题意。由于计算机是对 0 和 1 计算,因此我们计算的过程中会对一个数进行二进制(计算机底层运算)，十进制（go 语言默认进制），十六进制（题意要求）的转换。

因为 AES 算法中四个步骤之间存在联系，一个步骤的输出错了，就会影响下一个步骤。因此，我根据书本上的例题，编程过程中使用例题中的数值进行检验，保证编写的每一个步骤正确。

编写的代码存在的问题：有些步骤使用列举方式来编写，虽然易于理解但是代码冗余

本实验学到的内容：AES 算法的原理深入理解、锻炼编程思维、进制之间的转换关系、计算机底层二进制的计算

附件：程序代码

```
package main


import (
    "fmt"
    "strconv"
    "strings"
)


func main() {
    //定义我们的输入
    fmt.Println("(1)当输入为：32 43 f6 a8 88 5a 30 8d 31 31 98 a2 e0 37 07 34:")
    var arr01 = [16]byte{0x32, 0x43, 0xf6, 0xa8, 0x88, 0x5a, 0x30, 0x8d, 0x31, 0x31,
        0x98, 0xa2, 0xe0, 0x37, 0x07, 0x34}
    AES_94(arr01)
    fmt.Println()
    fmt.Println()


    fmt.Println("(2)当输入为：19 3d e3 be a0 f4 e2 2b 9a c6 8d 2a e9 f8 48 08:")
   var arr02 = [16]byte{0x19, 0x3d, 0xe3, 0xbe, 0xa0, 0xf4, 0xe2, 0x2b, 0x9a, 0xc6, 0x8d, 0x2a,
```

```
0xe9, 0xf8, 0x48, 0x08}
    AES_94(arr02)
    fmt.Println()
    fmt.Println()

    fmt.Println("(3)当输入为：20 21 13 10 94 00 00 00 00 00 00 00 00 00 00 0B:")
    var arr03 = [16]byte{0x20, 0x21, 0x13, 0x10, 0x94, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x0B}
    AES_94(arr03)

}

// 定义 S 盒
var Sbox = [16 * 16]byte{
    /*      0      1      2      3      4      5      6      7      8      9      a
b      c      d      e      f */
    /*0*/ 0x63, 0x7c, 0x77, 0x7b, 0xf2, 0x6b, 0x6f, 0xc5, 0x30, 0x01, 0x67, 0x2b, 0xfe, 0xd7,
0xab, 0x76,
    /*1*/ 0xca, 0x82, 0xc9, 0x7d, 0xfa, 0x59, 0x47, 0xf0, 0xad, 0xd4, 0xa2, 0xaf, 0x9c, 0xa4,
0x72, 0xc0,
    /*2*/ 0xb7, 0xfd, 0x93, 0x26, 0x36, 0x3f, 0xf7, 0xcc, 0x34, 0xa5, 0xe5, 0xf1, 0x71, 0xd8,
0x31, 0x15,
    /*3*/ 0x04, 0xc7, 0x23, 0xc3, 0x18, 0x96, 0x05, 0x9a, 0x07, 0x12, 0x80, 0xe2, 0xeb, 0x27,
0xb2, 0x75,
    /*4*/ 0x09, 0x83, 0x2c, 0x1a, 0x1b, 0x6e, 0x5a, 0xa0, 0x52, 0x3b, 0xd6, 0xb3, 0x29, 0xe3,
0x2f, 0x84,
    /*5*/ 0x53, 0xd1, 0x00, 0xed, 0x20, 0xfc, 0xb1, 0x5b, 0x6a, 0xcb, 0xbe, 0x39, 0x4a, 0x4c,
0x58, 0xcf,
    /*6*/ 0xd0, 0xef, 0xaa, 0xfb, 0x43, 0x4d, 0x33, 0x85, 0x45, 0xf9, 0x02, 0x7f, 0x50, 0x3c,
0x9f, 0xa8,
    /*7*/ 0x51, 0xa3, 0x40, 0x8f, 0x92, 0x9d, 0x38, 0xf5, 0xbc, 0xb6, 0xda, 0x21, 0x10, 0xff,
0xf3, 0xd2,
    /*8*/ 0xcd, 0x0c, 0x13, 0xec, 0x5f, 0x97, 0x44, 0x17, 0xc4, 0xa7, 0x7e, 0x3d, 0x64, 0x5d,
0x19, 0x73,
```

```
	/*9*/ 0x60, 0x81, 0x4f, 0xdc, 0x22, 0x2a, 0x90, 0x88, 0x46, 0xee, 0xb8, 0x14, 0xde, 0x5e, 0x0b, 0xdb,
	/*a*/ 0xe0, 0x32, 0x3a, 0x0a, 0x49, 0x06, 0x24, 0x5c, 0xc2, 0xd3, 0xac, 0x62, 0x91, 0x95, 0xe4, 0x79,
	/*b*/ 0xe7, 0xc8, 0x37, 0x6d, 0x8d, 0xd5, 0x4e, 0xa9, 0x6c, 0x56, 0xf4, 0xea, 0x65, 0x7a, 0xae, 0x08,
	/*c*/ 0xba, 0x78, 0x25, 0x2e, 0x1c, 0xa6, 0xb4, 0xc6, 0xe8, 0xdd, 0x74, 0x1f, 0x4b, 0xbd, 0x8b, 0x8a,
	/*d*/ 0x70, 0x3e, 0xb5, 0x66, 0x48, 0x03, 0xf6, 0x0e, 0x61, 0x35, 0x57, 0xb9, 0x86, 0xc1, 0x1d, 0x9e,
	/*e*/ 0xe1, 0xf8, 0x98, 0x11, 0x69, 0xd9, 0x8e, 0x94, 0x9b, 0x1e, 0x87, 0xe9, 0xce, 0x55, 0x28, 0xdf,
	/*f*/ 0x8c, 0xa1, 0x89, 0x0d, 0xbf, 0xe6, 0x42, 0x68, 0x41, 0x99, 0x2d, 0x0f, 0xb0, 0x54, 0xbb, 0x16}

// 字节代替
func subBytes_94(arr [16]byte) (newArr [16]byte) {
	for index, value := range arr {
		low := value & 0x0F		//取低 4 位
		high := (value >> 4) & 0x0f //取高 4 位
		newArr[index] = Sbox[16*high+low]
	}
	return newArr
}

// 行移位
func ShiftRows_94(arr [16]byte) (newArr [16]byte) {
	for index, _ := range arr {
		newArr[index] = arr[index]
	}
	//第一行不移位
	//第二行移位
	newArr[4] = arr[5]
	newArr[5] = arr[6]
```

```
    newArr[6] = arr[7]
    newArr[7] = arr[4]
    //第三行移位
    newArr[8] = arr[10]
    newArr[9] = arr[11]
    newArr[10] = arr[8]
    newArr[11] = arr[9]
    //第四行移位
    newArr[12] = arr[15]
    newArr[13] = arr[12]
    newArr[14] = arr[13]
    newArr[15] = arr[14]
    return newArr
}

// 列混合
func columnMix_94(arr [16]byte) [16]byte {
    //列混合之前先将数组重新排序（因为列混合是一列一列来的）
    var arrByte [16]byte = [16]byte{}
    arrByte[0] = arr[0]
    arrByte[1] = arr[4]
    arrByte[2] = arr[8]
    arrByte[3] = arr[12]
    arrByte[4] = arr[1]
    arrByte[5] = arr[5]
    arrByte[6] = arr[9]
    arrByte[7] = arr[13]
    arrByte[8] = arr[2]
    arrByte[9] = arr[6]
    arrByte[10] = arr[10]
    arrByte[11] = arr[14]
    arrByte[12] = arr[3]
    arrByte[13] = arr[7]
    arrByte[14] = arr[11]
```

```
	arrByte[15] = arr[15]

	result_ := [16]byte{} //用来装输出矩阵
	result_[0] = columnMixResult01_94(arrByte[0], arrByte[1], arrByte[2], arrByte[3])
	result_[1] = columnMixResult01_94(arrByte[4], arrByte[5], arrByte[6], arrByte[7])
	result_[2] = columnMixResult01_94(arrByte[8], arrByte[9], arrByte[10], arrByte[11])
	result_[3] = columnMixResult01_94(arrByte[12], arrByte[13], arrByte[14], arrByte[15])

	result_[4] = columnMixResult02_94(arrByte[0], arrByte[1], arrByte[2], arrByte[3])
	result_[5] = columnMixResult02_94(arrByte[4], arrByte[5], arrByte[6], arrByte[7])
	result_[6] = columnMixResult02_94(arrByte[8], arrByte[9], arrByte[10], arrByte[11])
	result_[7] = columnMixResult02_94(arrByte[12], arrByte[13], arrByte[14], arrByte[15])

	result_[8] = columnMixResult03_94(arrByte[0], arrByte[1], arrByte[2], arrByte[3])
	result_[9] = columnMixResult03_94(arrByte[4], arrByte[5], arrByte[6], arrByte[7])
	result_[10] = columnMixResult03_94(arrByte[8], arrByte[9], arrByte[10], arrByte[11])
	result_[11] = columnMixResult03_94(arrByte[12], arrByte[13], arrByte[14], arrByte[15])

	result_[12] = columnMixResult04_94(arrByte[0], arrByte[1], arrByte[2], arrByte[3])
	result_[13] = columnMixResult04_94(arrByte[4], arrByte[5], arrByte[6], arrByte[7])
	result_[14] = columnMixResult04_94(arrByte[8], arrByte[9], arrByte[10], arrByte[11])
	result_[15] = columnMixResult04_94(arrByte[12], arrByte[13], arrByte[14], arrByte[15])
	return result_
} //列混合主函数

func columnMixCount01_94(num byte) byte { //与{01}相乘
	return num
}

func columnMixCount02_94(num byte) byte { //与{02}相乘
	if num > 128 {
		return (num << 1) ^ 0b00011011
	} else {
		return num << 1
```

```
    }
}

func columnMixCount03_94(num byte) byte { //与{03}相乘
    return columnMixCount01_94(num) ^ columnMixCount02_94(num)
}

func columnMixResult01_94(num01 byte, num02 byte, num03 byte, num04 byte) byte { //固定矩阵第 1 行
    var s byte = columnMixCount02_94(num01) ^ columnMixCount03_94(num02) ^ columnMixCount01_94(num03) ^ columnMixCount01_94(num04)
    return s
}

func columnMixResult02_94(num01 byte, num02 byte, num03 byte, num04 byte) byte { //固定矩阵第 2 行
    var s byte = columnMixCount01_94(num01) ^ columnMixCount02_94(num02) ^ columnMixCount03_94(num03) ^ columnMixCount01_94(num04)
    return s
}

func columnMixResult03_94(num01 byte, num02 byte, num03 byte, num04 byte) byte { //固定矩阵第 3 行
    var s byte = columnMixCount01_94(num01) ^ columnMixCount01_94(num02) ^ columnMixCount02_94(num03) ^ columnMixCount03_94(num04)
    return s
}

func columnMixResult04_94(num01 byte, num02 byte, num03 byte, num04 byte) byte { //固定矩阵第 4 行
    var s byte = columnMixCount03_94(num01) ^ columnMixCount01_94(num02) ^ columnMixCount01_94(num03) ^ columnMixCount02_94(num04)
    return s
}
```

```
// 轮密钥加
func AddRoundKey_94(arr [16]byte) (newArr [16]byte) {
	//定义子密钥.
	var arrKey [16]byte = [16]byte{0xa0, 0x88, 0x23, 0x2a, 0xfa, 0x54, 0xa3, 0x6c, 0xfe, 0x2c,
0x39,
		0x76, 0x17, 0xb1, 0x39, 0x05}
	for i := 0; i < 16; i++ {
		newArr[i] = arrKey[i] ^ arr[i]
	}
	return newArr
}

func AES_94(arrBegin [16]byte) {
	step1 := subBytes_94(arrBegin)
	fmt.Print("字节代替:")
	for i := 0; i < 16; i++ {
		if i%4 == 0 {
			fmt.Println()
		}
		fmt.Printf("%3v", strings.ToUpper(strconv.FormatInt(int64(step1[i]), 16)))
	}
	fmt.Println()
	step2 := ShiftRows_94(step1)
	fmt.Print("行移位 :")
	for i := 0; i < 16; i++ {
		if i%4 == 0 {
			fmt.Println()
		}
		fmt.Printf("%3v", strings.ToUpper(strconv.FormatInt(int64(step2[i]), 16)))
	}
	fmt.Println()
	step3 := columnMix_94(step2)
	fmt.Print("列混合 :")
```

```
	for i := 0; i < 16; i++ {
		if i%4 == 0 {
			fmt.Println()
		}
		fmt.Printf("%3v", strings.ToUpper(strconv.FormatInt(int64(step3[i]), 16)))
	}
	fmt.Println()
	step4 := AddRoundKey_94(step3)
	fmt.Print("轮密钥加:")
	for i := 0; i < 16; i++ {
		if i%4 == 0 {
			fmt.Println()
		}
		fmt.Printf("%3v", strings.ToUpper(strconv.FormatInt(int64(step4[i]), 16)))
	}
}
```