**1.古典密码实验**

1.1替换技术

1.1.1凯撒加密

**实验原理**

把字母表中的每个字母向前循环移动3位

|  |
| --- |
| ABCDEFGHIJKLMNOPQRSTUVWXYZ |
| ↓ |
| DEFGHIJKLMNOPQRSTUVWXYZABC |

如：

k=3

ATTACKATFIVE-----------🡪DWWDFNDWILYH

改进：密文字母与明文字母不一定相隔三个字母，可以相隔任意多个字母，则可以提高破译的难度。密钥是1－25中其中的任一个数字这就是改进的凯撒加密方法。

**实验演示**





1.1.2维吉耐尔

**实验原理**

Vigenere（维吉耐尔）加密法是一种多码替换加密法。

密钥：

一个字符序列k=（k1,k2,……km）m为任意值

明文：

x=（x1,x2,……xn）将被分为长度为m的段，如果消息的长度恰好不是m的倍数，则最后一组的长度为小与m的段

加密函数：

解密函数：

例如：

使用YOUR为密钥，用Vigenere密码加密法加密明文HWARYO，则整加密过程为：

（1）明分为长4的段即对应如下：

P=HOWAREOU明）K=YOUR（密钥）

（2）加密，值表示密钥，则k=（25,15,21,18），m=4，第1个明文字符用其后面的第25个字符来代替，即向后移25位，第2个明文字符用其后面的第15个字符来代替，以此类推，最终加密的结果为：

Ek(P)=FCQRPSSFS

（3）解密：第一个密文字后用其前面第25个字符来代替，即向前移25位，第2个明文字符用其前面的第15个字符来代替，以此类推，最终获得解密结果。

**实验步骤**





1.1.3Vernam算法

**实验原理**

①按递增顺序把每个明文字母数字化，例如A=1，B=2，…Z=26，如表1.1所示。

表1.1字母数字化对照表

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 字母 | A | B | C | D | E | F | G | H | I | J | K | L | M |
| 对应的数字 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
| 字母 | N | O | P | Q | R | S | T | U | V | W | X | Y | Z |
| 对应的数字 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 |

②对密钥中每个字母进行相同处理。

③将明文中的每个字母与密钥中的相应字母相加，设和为s。

④如果s>26，执行赋值语句：s=smod26。

将s变成相应的字母，从而得到密文。

例如：用Vernam加密算法加密明文“ATTACKATFIVE”，密钥为“KSHUBGWMLVZX”。

加密后密文为：LMBVERXGREVC。

**实验步骤**





1.2置换加密技术

栅栏置换加密算法

**实验原理**

栅栏(RailFence)加密过程是：把要被加密的消息按照锯齿状一上一下的写法写出来。因为加密过程的几何形状类似于栅栏的上半部分，因此称为栅栏加密，如图1.1所示。

明文消息ATTACKATFIVE的加密过程

（1）把要被加密的消息按照锯齿状一上一下写出来：

A a

T

C

A

F

V

T

A

K

T

I

E

（2）然后一行一行地产生密文，得到ATCAFVTAKTIE

图1.1栅栏加密过程

解密过程是：先写第一行，再写第二行，每行字母的个数的决定是按照下面的原则进行的：字母总数是偶数时第一行和第二行各一半，奇数时第一行多一个。然后按加密对角线序列读出。

**实验步骤**





单轮分栏式置换加密技术

**实验原理**

**单轮分栏式置换加密过程**

将明文消息一行一行地写入预定长度的矩形中(需要事先确定列数)。然后一列一列读消息，但不一定按1、2、3列的自然顺序读，也可以按随机顺序读，得到的消息就是密文消息。

**例：**明文为“begantoattackattwo”，共6列，加密时按6,5,4,3,2,1列读出，得密文为：tconawattgtteaabok，如下表所示。



**单轮分栏式置换解密过程**

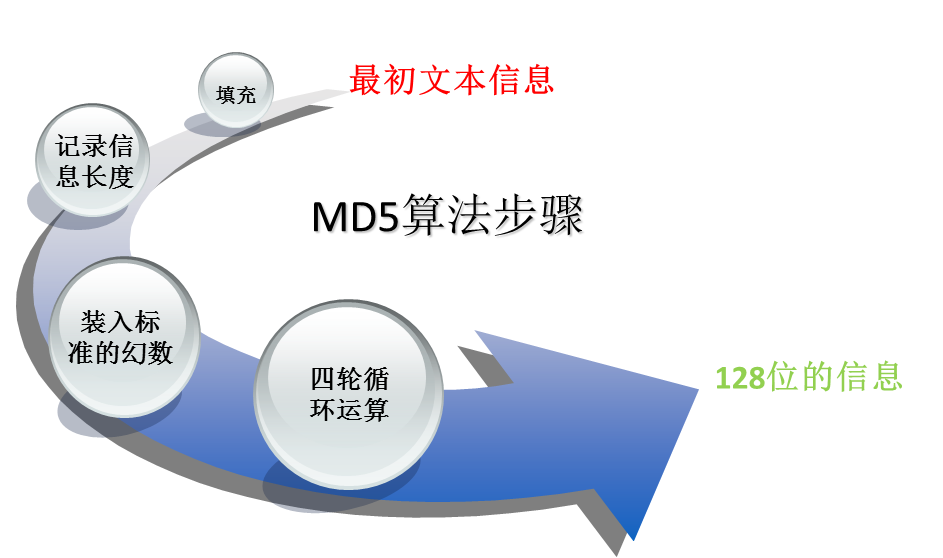
根据密文字母的总个数（被除数）和列数（除数）确定行数（商）和最后一行非空白单元的个数（余数），例如，设总字母个数为16，共6列，则整行数为：，第三行非空白单元为16mod6=4，按读列的顺序写入矩形中，然后一行一行地读出，就得到明文。

**实验步骤**

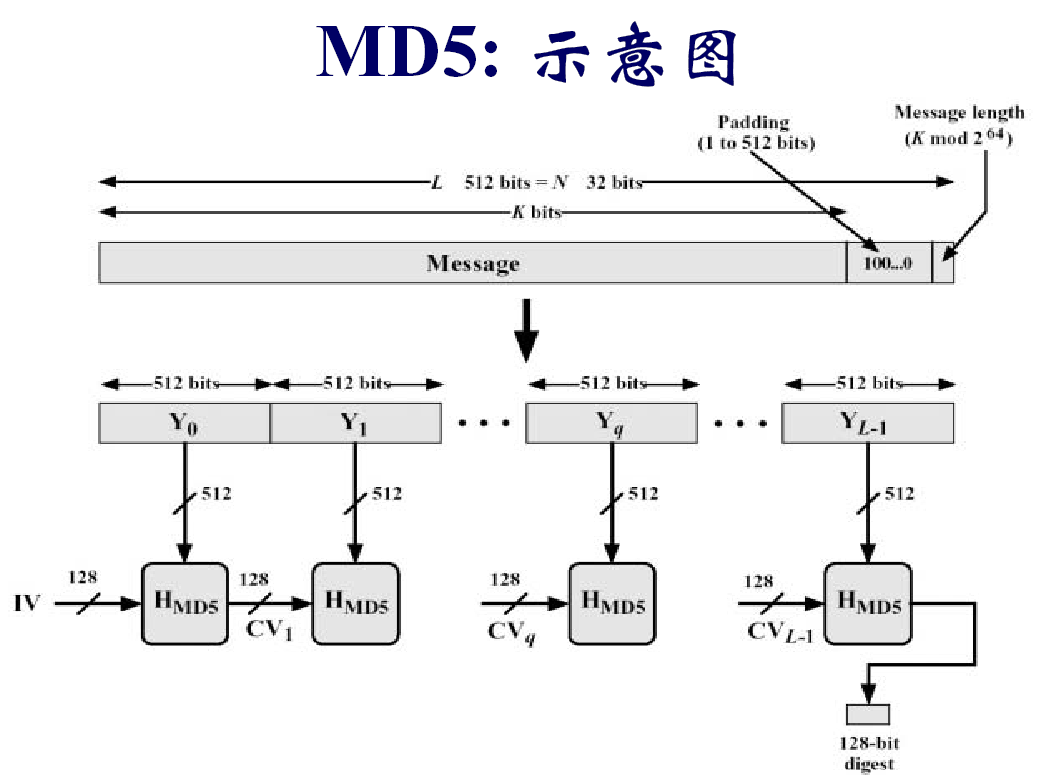
**2.hash算法实验**

2.1MD5

**实验原理**



* 输入：任意长度的报文
* 输入分组长度：512bit
* 输出：128bit报文



**MD5描述－step1:填充**

* 对报文进行填充，使其比特数与448模512同余，即填充长度为512的整数倍减去64



* 填充方法：填充比特串的最高位为1，其余各位均为0

**MD5描述－step2:添加长度**

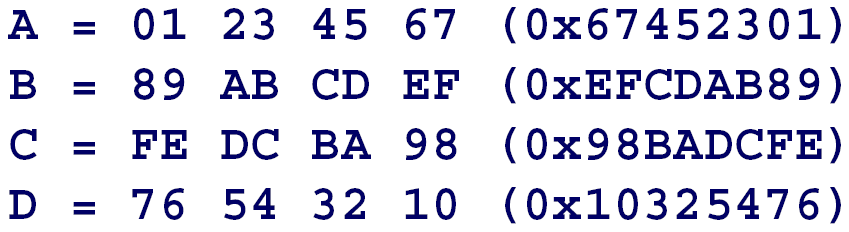
* 附加长度值



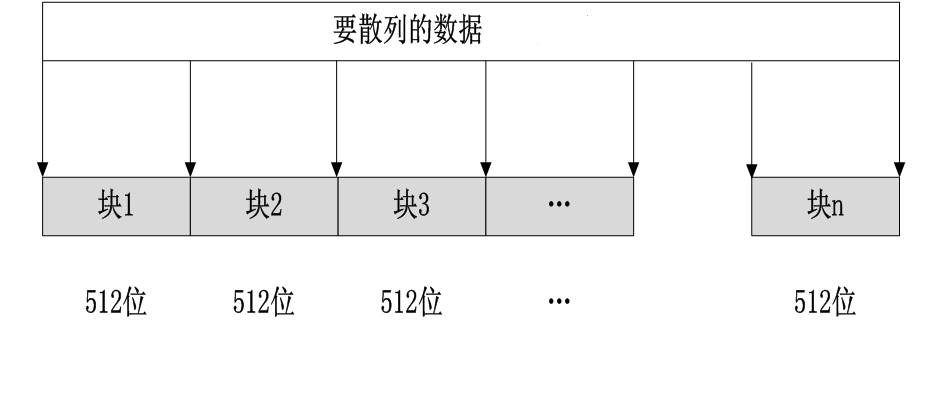
* |M2|为512的倍数:Y0,Y1,…,YL-1

**MD5描述－step3:初始化链接变量**

* MD为128bit，用于存放散列函数的中间及最终结果
* MD可表示为4个32bit的寄存器(A,B,C,D)，初始化如下



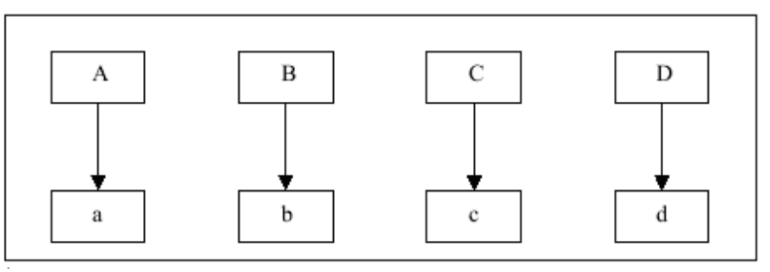
**MD5描述－step4:将输入分成512位的块**

****

**MD5描述－step5**

* 初始化之后,就要开始实际算法了,这是个循环,对消息中的多个512位块进行.

1、将四个链接变量复制到四个变量中



实际上，这个算法将a、b、c、d组合成128位寄存器(abcd)，寄存器(abcd)在实际算法运算中保存中间结果和最终结果。

2、将当前512位块分解为16个子块，每个子块为32位

3、主循环有四轮,每轮很相似，每一轮的操作,都要处理一个块中的16个子块。

* 每一轮（16次迭代）的输入如下：

(a)16个子块(32位)M[0]，M[1]，…，M[15]

(b)变量a,b,c,d（每个32位）

(c)常量t（16/64个，每个32位）

表示为t[k]，其中k为1～64。由于有四轮，因此每一轮用64个t值中的16个

(d)S(循环左移S位)

* 这四轮中的第1步进行不同处理，其他步骤是相同的。下面总结这四轮的迭代。每一轮输出的中间和最终结果复制到寄存器abcd中，注意，每一轮有16个寄存器。

1)首先对b,c,d作一次非线性函数运算，这个运算在四轮中不同。

2)变量a加进第1步的输出(即寄存器abcd)。

3)消息子块M[i]加进第2步的输出(即寄存器abcd)。

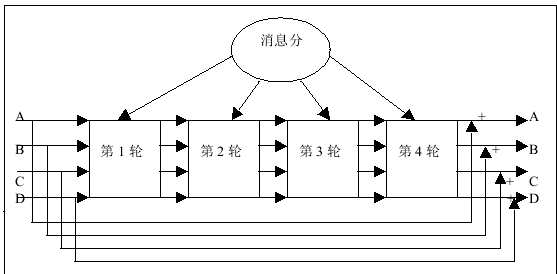
4)常量t[i]加进第3步输出(即寄存器abcd)。

5)第4步的输出(即寄存器abcd)循环左移s位。

6)变量b加进第5步输出(即寄存器abcd)。

7)第6步的输出成为下一步的新abcd。

MD5的操作过程下图



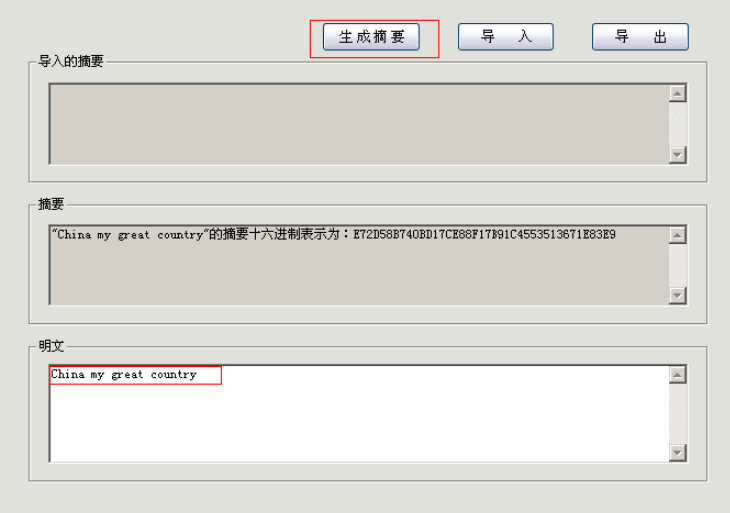
MD5主循环



每一轮的操作

所有这些完成之后，将A，B，C，D分别加上a，b，c，d，然后用下一分组数据继续运行算法，最后MD5算法产生128位的输出是A，B，C和D的级联，其中低字节始于A，高字节终于D。至此整个MD5算法处理结束。

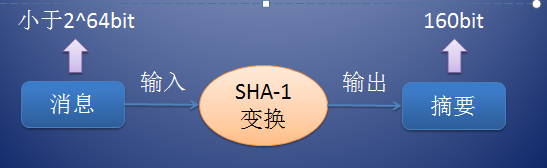
**实验演示**



2.2SHA

**实验原理**

* 可以对长度不超过2^64比特的消息进行计算，输入以512位数据块为单位处理，产生160比特的消息摘要作为输出。



* 准备工作：把原始消息（如字符串、文件等）转换成bit序列。

“abc”

011000010110001001100011

原始消息

bit序列

转换

‘a’=0x61‘b’=0x62‘c’=0x63

* 步骤1：补位。最高位补1，其余补0，使得数据位总长度对512取余的结果为448。

011000010110001001100011

原始序列

补位后序列

011000010110001001100011

10000000000……00000000000

24位

补1位1，补423位0

448位

* 补位。最高位补1，其余补0，使得数据位总长度对512取余的结果为448。

**61626380000000000000000000000000**

**00000000000000000000000000000000**

**00000000000000000000000000000000**

**0000000000000000**

补位后的序列

转换成16进制

* 步骤2：补长度。将原始数据的长度补到补位操作后的消息后面。

**61626380000000000000000000000000**

**00000000000000000000000000000000**

**00000000000000000000000000000000**

00000000000000000000000000000018

补位后的序列

转换成16进制

* 步骤3：将步骤2得到的消息M以512bit分组。

M1

**M=M1||M2||…||Mi||…||Mn**

**512bit**

**512bit**

**512bit**

**512bit**

**M=61626380000000000000000000000000 00000000000000000000000000000000 00000000000000000000000000000000**

**00000000000000000000000000000018**

* 步骤4：对于Mi进行以下处理
* （1）将Mi分成16个字W0，W1，…,W15

M1=61626380000000000000000000000000

W0W1W2W3

00000000000000000000000000000000

W4W5W6W7

00000000000000000000000000000000

W8W9W10W11

00000000000000000000000000000018

W12W13W14W15

* （2）将16个字扩展至80个字,W16，W17，…,W79

Wt=S1(Wt-3⊕Wt-8⊕Wt-14⊕Wt-16)(0≤t≤19)

St(W)将W循环左移t位

* 步骤5：进行散列函数的计算,SHA-1算法的核心共4轮，每轮20步，共80步。
* （1）初始化MD缓冲区{Hi}，160bit(5个32位的寄存器)。

H0=0x67452301

H1=0xEFCDAB89

H2=0x98BADCFE

H3=0x10325476

H4=0xC3D2E1F0

* (2)更新缓存ABCDE的内容，缓存ABCDE的初始值为：

A=H0

B=H1

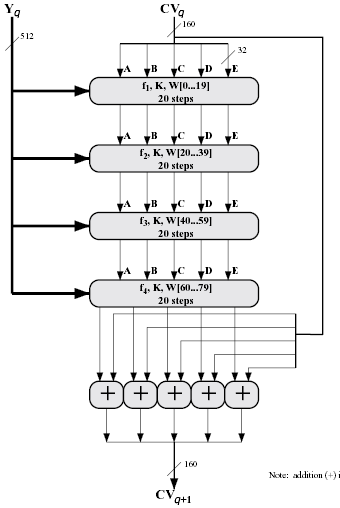
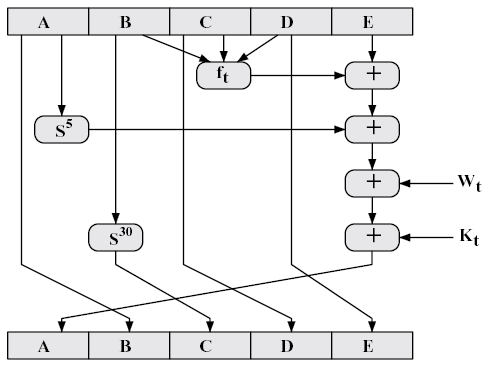
C=H2

D=H3

E=H4

* 共4轮，每轮的常数K为：

Kt=5A827999(0≤t≤19)   
Kt=6ED9EBA1(20≤t≤39)   
Kt=8F1BBCDC(40≤t≤59)   
Kt=CA62C1D6(60≤t≤79)

* 
* 
* （3）对于0≤t≤79，执行以下循环：

**TEMP=S5(A)+ft(B,C,D)+E+Wt+Kt**

**ft(B,C,D)=(B∧C)∨((¬B)∧D)(0≤t≤19)   
ft(B,C,D)=B⊕C⊕D(20≤t≤39)   
ft(B,C,D)=(B∧C)∨(B∧D)∨(C∧D)(40≤t≤59)   
ft(B,C,D)=B⊕C⊕D(60≤t≤79)**

**B=A，C=(B<<30)，D=C，E=D**

E=D;D=C;C=S30(B);B=A;A=TMP

* （4）上一步循环执行完成后，令

H0=H0+A

H1=H1+B

H2=H2+C

H3=H3+D

H4=H4+E

* 步骤6：重复步骤5，对消息块M1…Mn进行处理。

TEMP=S5(A)+ft(B,C,D)+E+Wt+Kt

E=D;D=C;C=S30(B);B=A;A=TMP

* 处理完所有的Mn后，得到160bit消息摘要

H0H1H2H3H4

**实验演示**



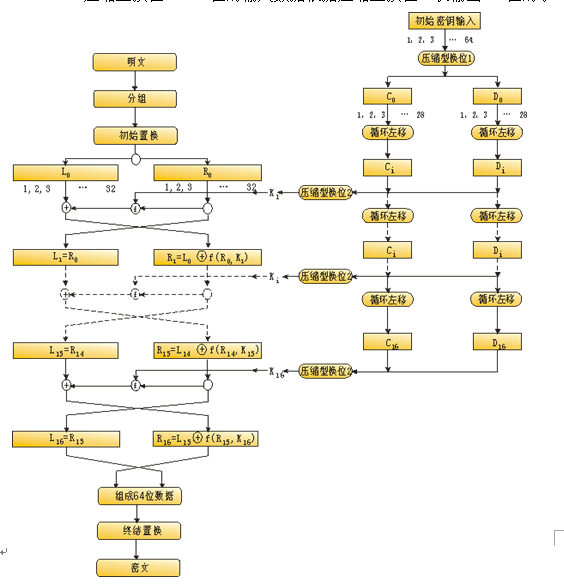
**3.分组密码实验**

DES

**实验原理**

主要步骤:

1. 输入64位明文
2. 将64位明文块送入初始设置函数进行初始置换
3. 将初始置换后的内容分为两块：左明文L0和右明文R0
4. 每个左明文与右明文经过16轮的加密过程
5. 将左明文和右明文重接起来，对组成的块进行最终置换
6. 输出64位密文



**实验演示**

对字符进行加密

对文件进行加密

**4.非对称密码实验**

RSA算法

**实验原理**

* 算法基础：
  + RSA算法基于这样的数学事实：两个大素数很容易相乘，而对得到的积求因子则很难。
* RSA中的私钥和公钥基于大素数(100位以上)

RSA密钥的计算

* 首先选取一个整数N，要求N的值很大，而且是一对大素数P和Q的乘积。P和Q值一般要求在100位以上，这样得到的N值就有200位以上，如此的大数可以有效阻止对N的因子分解。
* 接着选择一个较大的整数E（公钥），使得E不是(P-1)×(Q-1)的因子，方法是先求因子，再选择非因子的数，选取的结果可能不唯一。
* 选择一个整数D（私钥），使得E与D之积E×D关于(P-1)×(Q-1)的余数为1，即存在：

E×Dmod(P-1)×(Q-1)＝1

选取的结果也不唯一，为了加快计算D的速度，可选取较小的。

* 这样就得到了一对密钥(E，N)和(D，N)，注意RSA中应对P、Q和D进行保密，事实上，P与Q在生成密钥后即被销毁。
* 加密算法：设明文为PT，密文为CT，

则有CT＝PTEmodN。

* 解密算法：设密文为CT，明文为PT，

则有PT＝CTDmodN。

（1）选择两个大素数P、Q。

（2）计算N=P×Q。

（3）选择一个公钥(即加密密钥)E，使其不是(P-1)与(Q-1)的因子

（4）选择私钥(即解密密钥)D，满足下列条件：

(D×E)mod(P—1)×(Q—1)＝1

(5）加密时，从明文PT计算密文CT如下：

CT＝PTEmodN

（6）将密文CT发送给接收方。

（7）解密时，从密文CT计算明文PT如下：

PT＝CTDmodN

**实验演示**

对某个字符串进行加密

**5.对称密码实验**

AES算法

**实验原理**



**加密过程基本变换**

(1)S盒变换

S盒变换是一个基于S盒的非线性置换，它用于输入或中间态的每一个字节通过一个简单的查表操作，将其映射为另一个字节。映射方法是：把输入字节的高4位作为S盒的行值，低4位作为列值，然后取出S盒中对应行和列的元素作为输出。例如，输入为“89”(十六进制)的值所对应的S盒的行值为“8”，列值为“9”，S盒中相应位置的值为“a7”，就说明“89”被映射为“87”。

(2)列混淆变换

列混淆变换实现逐列混淆，其方法是：



其中，，{x}内的数表示是字节。

用矩阵表示为



(3)行移位变换

行移位变换完成基于行的循环移位操作，即行移位变换的作用在中间态的行上，第0行不动，第1行循环左移1个字节，第2行循环左移2个字节，第3行循环左移3个字节。

(4)轮密钥加变换

轮密钥加变换用于将输入或中间态S的每一列与一个密钥字W[i]进行按位异或，其中，由原始密钥通过密钥扩展算法产生。

**解密过程基本变换**

解密过程是加密的逆过程，S盒变换、行移位变换、列混淆变换都要进行求逆变换，即逆S盒变换、逆行移位变换、逆列混淆变换。而轮密钥加变换与加密过程相同。

(1)逆S盒变换

与S盒变换类似，逆S盒变换是基于逆S盒实现的。

(2)逆行移位变换

与行移位变换相反，逆行移位变换将态State的后三行按相反的方向进行移位操作，即第0行保持不变，第1行向右移1个字节，第2行向右移2个字节，第3行向右移3个字节。

(3)逆列混淆变换

逆列混淆变换的处理方法与列混淆变换类似，每一列都通过与一个固定的多项相乘进行变换。

写为矩阵乘法的形式，逆列混淆变换按以下方式对列进行变换：



**实验演示**



0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31

<template>

<div class="app-container">

<el-tabs>

<el-tab-pane label="算法原理">

<h3 class="title" v-text="$route.meta.name"></h3>

<div class="img-wrap">

<img src="../../assets/img/aes.jpg" ondragstart="return false" alt="实验原理">

</div>

</el-tab-pane>

<el-tab-pane label="算法演示">

<h3 class="title" v-text="$route.meta.name"></h3>

<div class="calc-demo">

<el-form label-position="right" label-width="80px" :model="form">

<el-form-item label="文本">

<el-input v-model="form.text" placeholder="请输入加密文本"></el-input>

</el-form-item>

<el-form-item label="秘钥">

<el-input v-model="form.key" placeholder="请输入加密秘钥"></el-input>

</el-form-item>

<el-form-item label="加密">

<el-input type="textarea" :rows="2" v-model="form.password" :disabled="true"></el-input>

</el-form-item>

<el-form-item label="方式">

<div>

<el-radio v-model="form.type" style="margin-left: 10px;" label="128bits" size="medium" border>128bits

</el-radio>

<el-radio v-model="form.type" label="192bits" size="medium" border>192bits</el-radio>

<el-radio v-model="form.type" label="256bits" size="medium" border>256bits</el-radio>

</div>

</el-form-item>

<el-form-item label="操作">

<el-switch

v-model="form.control"

active-text="加密"

inactive-text="解密">

</el-switch>

</el-form-item>

<el-button class="bt-jm" @click="getAes" type="primary">加密</el-button>

</el-form>

</div>

</el-tab-pane>

</el-tabs>

</div>

</template>

<script>

import aesjs from 'aes-js';

export default {

name: "index",

data() {

return {

form: {

text: '华北科技学院',

password: '',

type: '128bits',

control: true,

key: '0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31'

}

}

},

methods: {

getAes() {

if (this.form.text.trim() !== "" && this.form.key.trim() !== "") {

let key;

const pw = this.form.key;

if (this.form.type === '128bits') {

if (pw.length < 16) {

this.$notify({

title: '警告',

message: '128bits加密,秘钥最少为16位!',

type: 'warning'

});

return;

}else{

key = pw.split(',').slice(0,16);

}

} else if (this.form.type === '192bits') {

if (pw.length < 24) {

this.$notify({

title: '警告',

message: '192bits加密,秘钥最少为24位!',

type: 'warning'

});

return;

}else{

key = pw.split(',').slice(0,24);

}

} else {

if (pw.length < 32) {

this.$notify({

title: '警告',

message: '256bits加密,秘钥最少为32位!',

type: 'warning'

});

return;

}else{

key = pw.split(',').slice(0,32);

}

}

debugger

console.log(key)

key = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]

console.log(key)

if (this.form.control === true) {

// 加密

let text = this.form.text;

const textBytes = aesjs.utils.utf8.toBytes(text);

let aesCtr = new aesjs.ModeOfOperation.ctr(key, new aesjs.Counter(5));

let encryptedBytes = aesCtr.encrypt(textBytes);

this.form.password = aesjs.utils.hex.fromBytes(encryptedBytes);

} else {

// 解密

let encryptedHex = this.form.text;

let encryptedBytes = aesjs.utils.hex.toBytes(encryptedHex);

let aesCtr = new aesjs.ModeOfOperation.ctr(key, new aesjs.Counter(5));

let decryptedBytes = aesCtr.decrypt(encryptedBytes);

this.form.password = aesjs.utils.utf8.fromBytes(decryptedBytes);

}

} else {

this.form.password = "";

this.$notify({

title: '警告',

message: '请填写秘钥和加密文本!',

type: 'warning'

});

}

}

}

}

</script>

<style scoped>

.bt-jm {

margin: 0 auto;

display: block;

}

</style>

<template>

<div class="app-container">

<el-tabs>

<el-tab-pane label="算法原理">

<h3 class="title" v-text="$route.meta.name"></h3>

<div class="img-wrap">

<img src="../../assets/img/aes.jpg" ondragstart="return false" alt="实验原理">

</div>

</el-tab-pane>

<el-tab-pane label="算法演示">

<h3 class="title" v-text="$route.meta.name"></h3>

<div class="calc-demo">

<el-form label-position="right" label-width="80px" :model="form">

<el-form-item label="文本">

<el-input v-model="form.text" placeholder="请输入加密文本"></el-input>

</el-form-item>

<el-form-item label="秘钥">

<el-input v-model="form.key" placeholder="请输入加密秘钥"></el-input>

</el-form-item>

<el-form-item label="加密">

<el-input type="textarea" :rows="2" v-model="form.password" :disabled="true"></el-input>

</el-form-item>

<el-form-item label="方式">

<div>

<el-radio v-model="form.type" style="margin-left: 10px;" label="128bits" size="medium" border>128bits

</el-radio>

<el-radio v-model="form.type" label="192bits" size="medium" border>192bits</el-radio>

<el-radio v-model="form.type" label="256bits" size="medium" border>256bits</el-radio>

</div>

</el-form-item>

<el-form-item label="操作">

<el-switch

v-model="form.control"

active-text="加密"

inactive-text="解密">

</el-switch>

</el-form-item>

<el-button class="bt-jm" @click="getAes" type="primary">加密</el-button>

</el-form>

</div>

</el-tab-pane>

</el-tabs>

</div>

</template>

<script>

import aesjs from 'aes-js';

export default {

name: "index",

data() {

return {

form: {

text: '华北科技学院',

password: '',

type: '128bits',

control: true,

key: '0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31'

}

}

},

methods: {

getAes() {

if (this.form.text.trim() !== "" && this.form.key.trim() !== "") {

let key = [];

let pw = this.form.key.split(',');

if (this.form.type === '128bits') {

if (pw.length < 16) {

this.$notify({

title: '警告',

message: '128bits加密,秘钥为16位!',

type: 'warning'

});

return;

} else {

for (let i = 0; i < 16; i++) {

key.push(parseInt(pw[i]))

}

}

} else if (this.form.type === '192bits') {

if (pw.length < 24) {

this.$notify({

title: '警告',

message: '192bits加密,秘钥为24位!',

type: 'warning'

});

return;

} else {

for (let i = 0; i < 24; i++) {

key.push(parseInt(pw[i]))

}

// key = pw.split(',').slice(0, 24);

}

} else {

if (pw.length < 32) {

this.$notify({

title: '警告',

message: '256bits加密,秘钥为32位!',

type: 'warning'

});

return;

} else {

for (let i = 0; i < 32; i++) {

key.push(parseInt(pw[i]))

}

// key = pw.split(',').slice(0, 32);

}

}

// console.log(key)

// key = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]

// console.log(key)

if (this.form.control === true) {

// 加密

let text = this.form.text;

const textBytes = aesjs.utils.utf8.toBytes(text);

let aesCtr = new aesjs.ModeOfOperation.ctr(key, new aesjs.Counter(5));

let encryptedBytes = aesCtr.encrypt(textBytes);

this.form.password = aesjs.utils.hex.fromBytes(encryptedBytes);

} else {

// 解密

let encryptedHex = this.form.text;

let encryptedBytes = aesjs.utils.hex.toBytes(encryptedHex);

let aesCtr = new aesjs.ModeOfOperation.ctr(key, new aesjs.Counter(5));

let decryptedBytes = aesCtr.decrypt(encryptedBytes);

this.form.password = aesjs.utils.utf8.fromBytes(decryptedBytes);

}

} else {

this.form.password = "";

this.$notify({

title: '警告',

message: '请填写秘钥和加密文本!',

type: 'warning'

});

}

}

}

}

</script>

<style scoped>

.bt-jm {

margin: 0 auto;

display: block;

}

</style>