项目完成了DES、keeloq、Playfir、RSA、仿射密码、Vigenere、Hill（py版本），并使用图形化界面将各个部分串联起来。

以下附上一些个人对于这些密码算法的理解与看法：

DES是分组密码，keeloq是序列密码，都属于对称密码体制

RSA是非对称密码

仿射密码、Playfir、Vigenere、Hill采用的是古典密码中的替换密码技术（仿射密码是单字符单表替换Hill、Playfir和Vigenere是单字符多表替换）

**对称密码体制与非对称密码体制**

对称加密与解密采用的是同样的密钥加密，解密的速度比较快，适合数据比较长时的使用。但是密钥传输的过程不安全，且容易被破解，密钥管理也比较麻烦。

而非对称密码的加解密使用的密钥是不一样的，使用公钥加密、私钥解密，这一点安全性更高了,但是加密和解密花费的时间较长，只适合对少量的数据进行加密。

**序列密码和分组密码**

分组密码(块加密)一次加密明文中的一个块，明文按照一定位长分组，明文组经过加密运算得到密文组，密文组经过解密运算（加密运算的逆运算），还原成明文组。

序列密码(流加密)一次加密明文中的一个位，利用少量的密钥（制乱元素）通过复杂的密码算法运算产生大量的伪随机位流，用于对明文位流的加密。解密用同样的密钥和密码算法及与加密相同的伪随机位流，用以还原明文位流。

**分组加密四种工作模式**

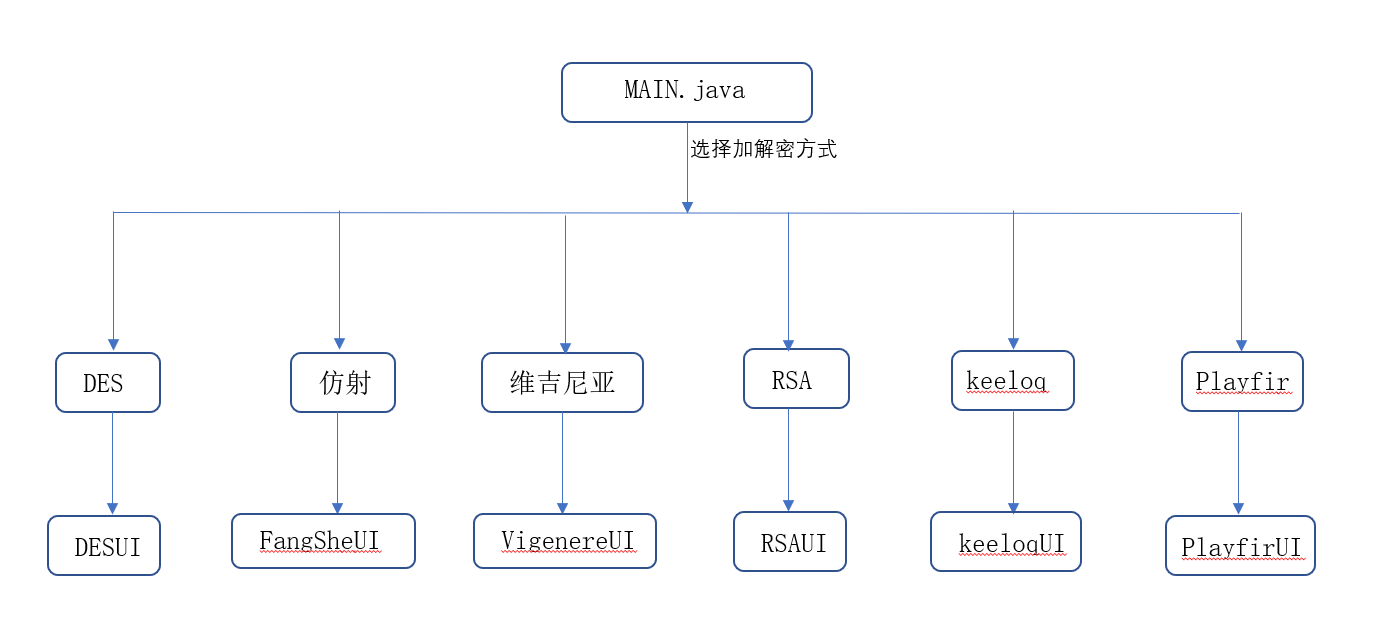
ECB(电子编码本)：明文消息被分成固定大小的块使用相同的方法独立进行加解密，所以可以进行并行计算，但是一旦有一个块被破解，使用相同的方法可以解密所有的明文数据，安全性比较差。适用于数据较少的情形，加密前需要把明文数据填充到块大小的整倍数。

**CBC (密码分组链接)：**每一个分组要先和前一个分组加密后的数据进行异或，然后再进行加密。这样每个密文块依赖该块之前的所有明文块，因此不能并行处理。为了保持每条消息都具有唯一性，第一个数据块进行加密之前需要用初始化向量IV进行异或。明文块必须填充到块大小的整倍数。

**CFB (密码反馈)：**和CBC模式比较相似，前一个分组的密文加密后和当前分组的明文XOR异或操作生成当前分组的密文。CFB模式的解密和CBC模式的加密在流程上其实是非常相似的。

**OFB (输出反馈)：**将分组密码转换为同步流密码，可以根据明文长度先独立生成相应长度的流密码。前一个分组与前一个明文块异或之前的流密码XOR当前分组明文，和CFB非常相似。由于异或操作的对称性，OFB模式的解密和加密完全一样的流程。

**图形化界面设计思路**

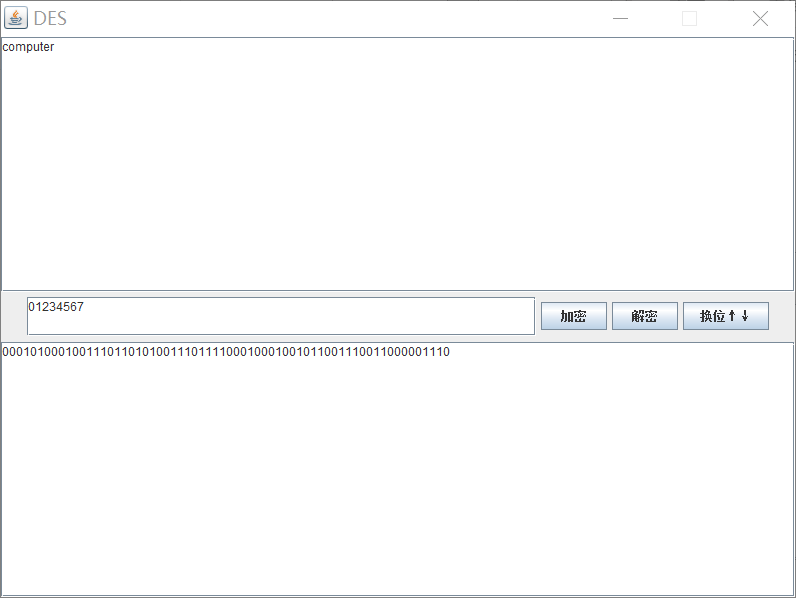


**各个算法界面**

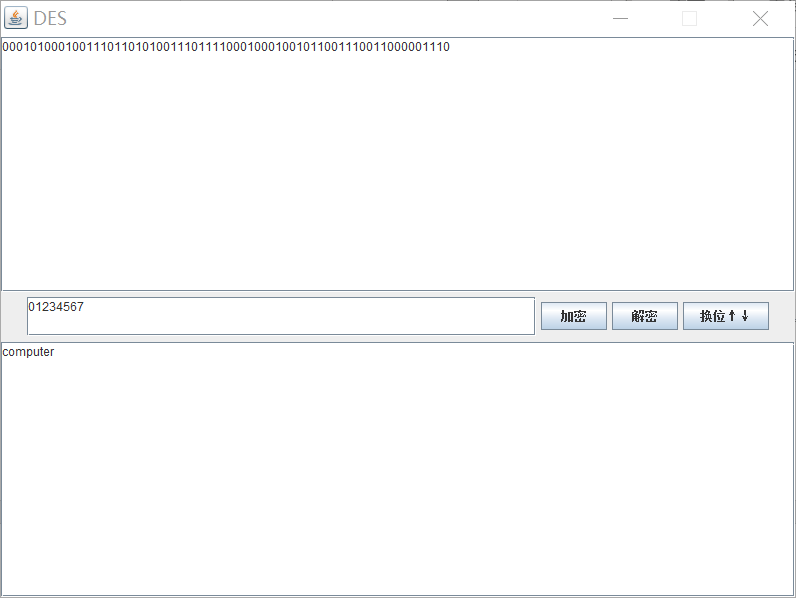


**DES模式:**

**加密** 明文：computer 密钥：01234567 -->密文二进制串：0001010001001110110101001110111100010001001011001110011000001110

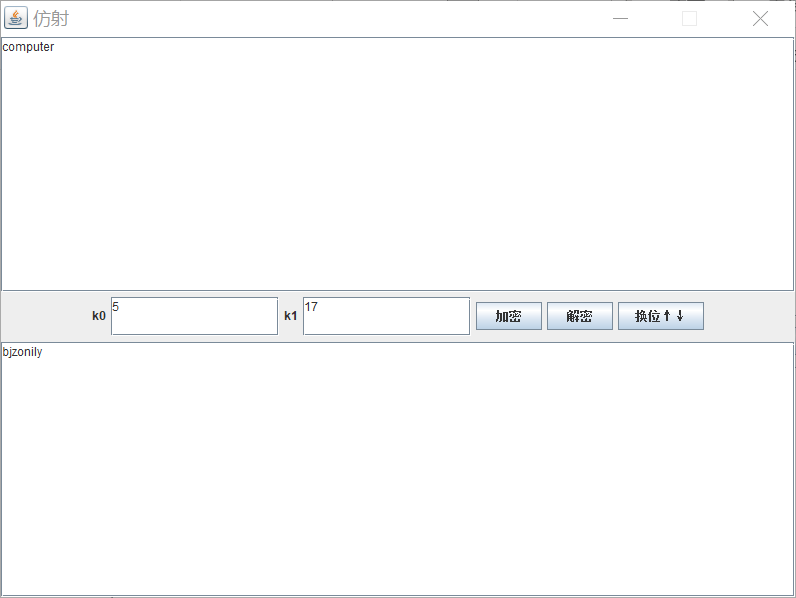


**解密** 密文二进制串：0001010001001110110101001110111100010001001011001110011000001110密钥：01234567 -->明文：computer

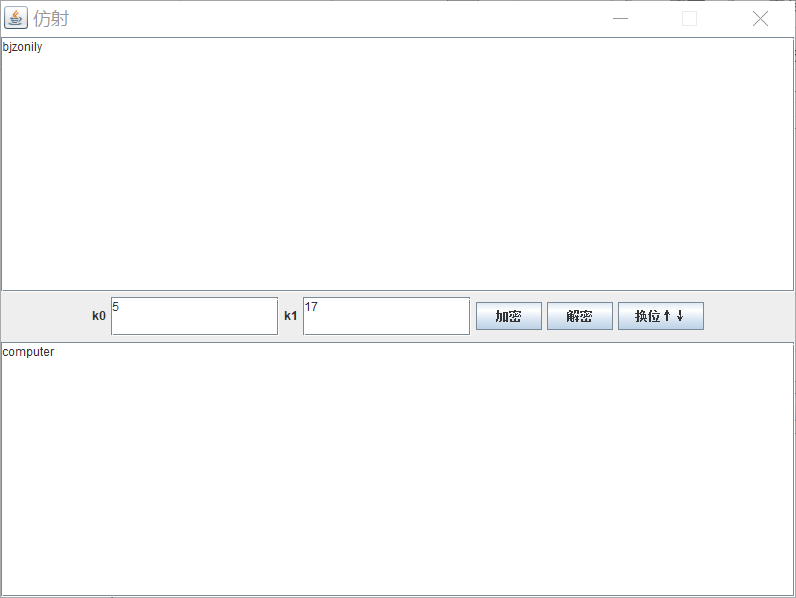


**仿射**

**加密** 明文：computer k0：5 k1：17 -->密文：bjzonily

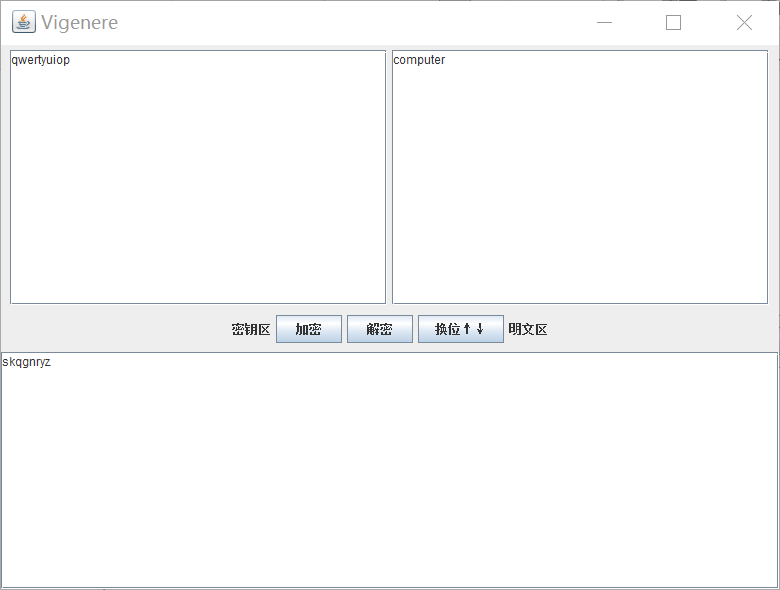


**解密** 密文：bjzonily k0：5 k1：17 -->密文：computer

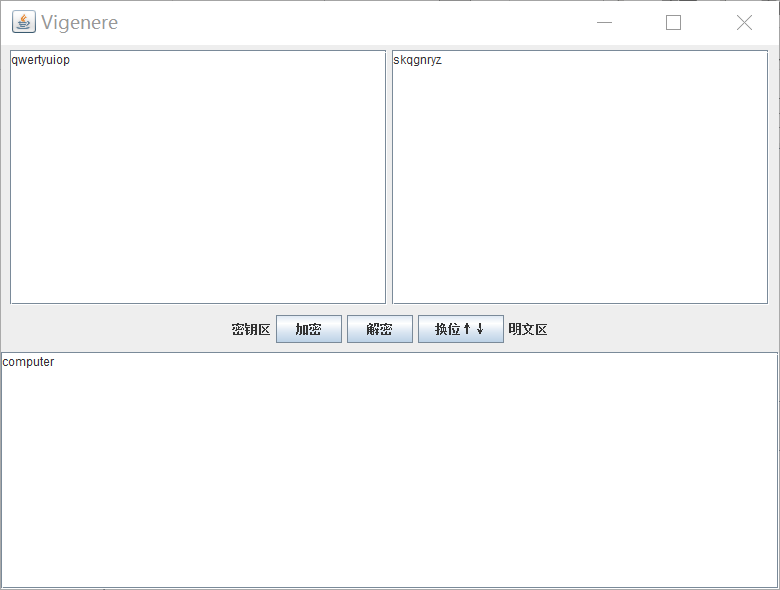


**维吉尼亚**

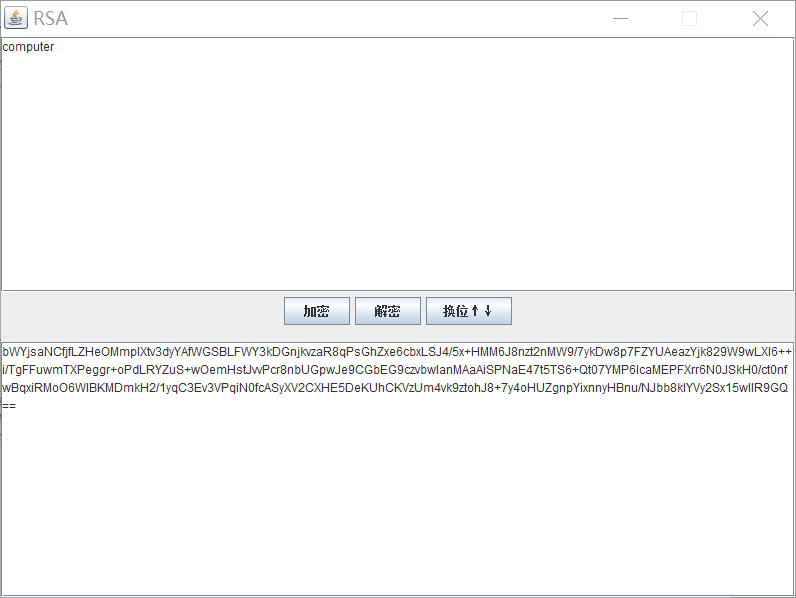
**加密** 明文：computer 密钥：qwertyuiop -->密文：skqgnryz



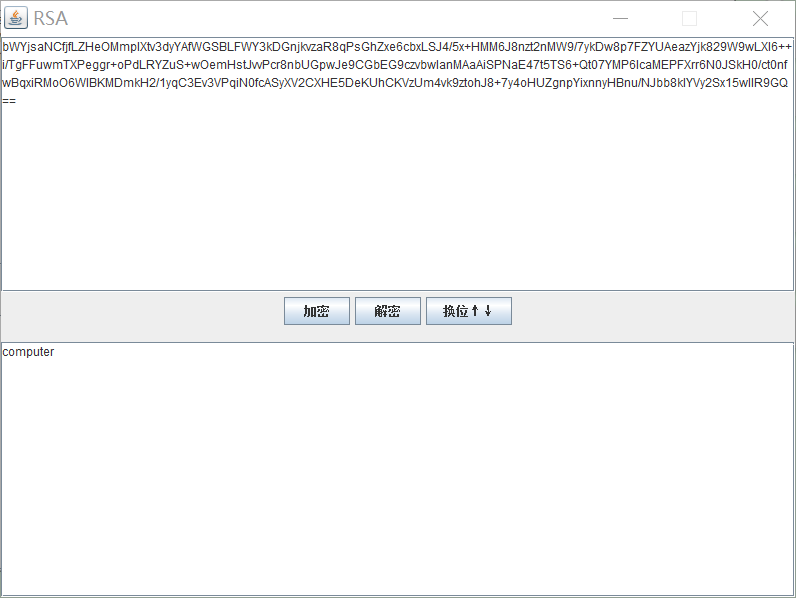
**解密** 密文：skqgnryz 密钥：qwertyuiop -->密文：computer



**RSA**

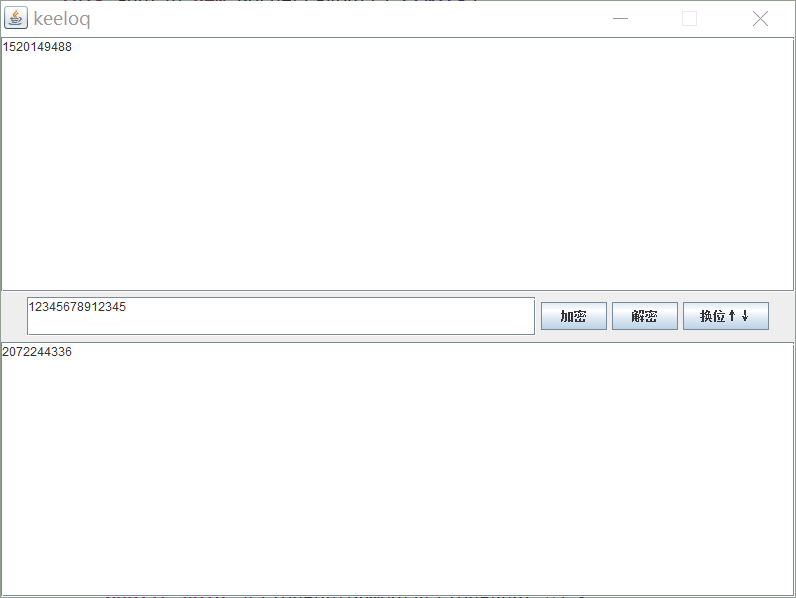
**加密** 明文：computer -->密文base64进制字符串：bWYjsaNCfjfLZHeOMmplXtv3dyYAfWGSBLFWY3kDGnjkvzaR8qPsGhZxe6cbxLSJ4/5x+HMM6J8nzt2nMW9/7ykDw8p7FZYUAeazYjk829W9wLXl6++i/TgFFuwmTXPeggr+oPdLRYZuS+wOemHstJvvPcr8nbUGpwJe9CGbEG9czvbwIanMAaAiSPNaE47t5TS6+Qt07YMP6IcaMEPFXrr6N0JSkH0/ct0nfwBqxiRMoO6WlBKMDmkH2/1yqC3Ev3VPqiN0fcASyXV2CXHE5DeKUhCKVzUm4vk9ztohJ8+7y4oHUZgnpYixnnyHBnu/NJbb8klYVy2Sx15wllR9GQ==

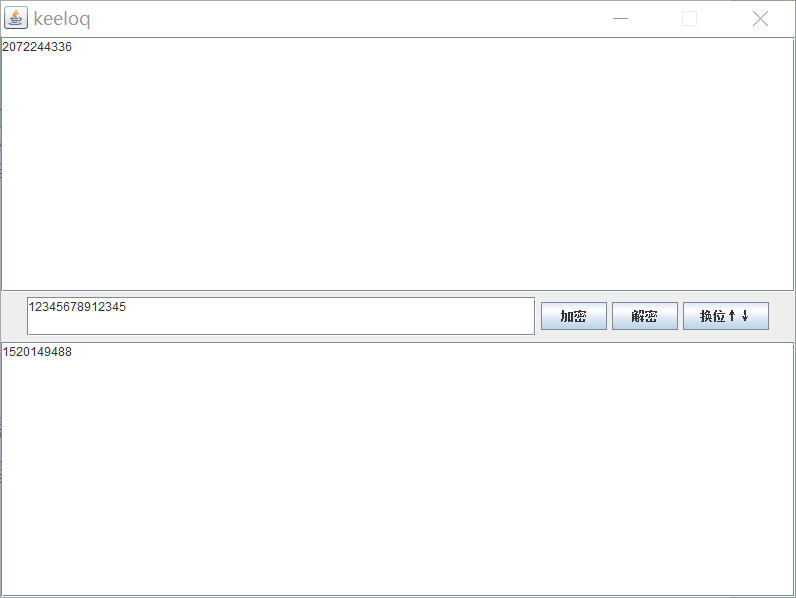
**解密** 密文base64进制字符串：bWYjsaNCfjfLZHeOMmplXtv3dyYAfWGSBLFWY3kDGnjkvzaR8qPsGhZxe6cbxLSJ4/5x+HMM6J8nzt2nMW9/7ykDw8p7FZYUAeazYjk829W9wLXl6++i/TgFFuwmTXPeggr+oPdLRYZuS+wOemHstJvvPcr8nbUGpwJe9CGbEG9czvbwIanMAaAiSPNaE47t5TS6+Qt07YMP6IcaMEPFXrr6N0JSkH0/ct0nfwBqxiRMoO6WlBKMDmkH2/1yqC3Ev3VPqiN0fcASyXV2CXHE5DeKUhCKVzUm4vk9ztohJ8+7y4oHUZgnpYixnnyHBnu/NJbb8klYVy2Sx15wllR9GQ== --> 明文：computer



**Keeloq**

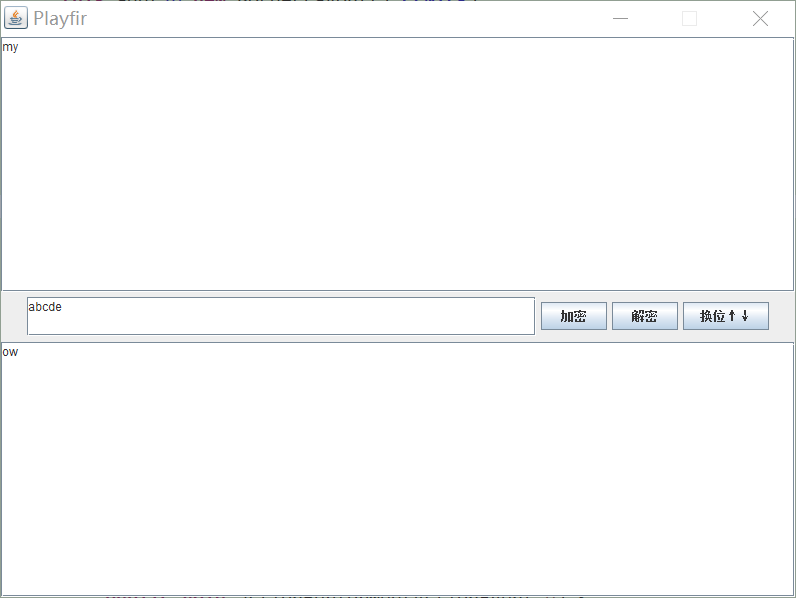
**加密** 明文：1520149488 密钥：12345678912345-->密文：2072244336



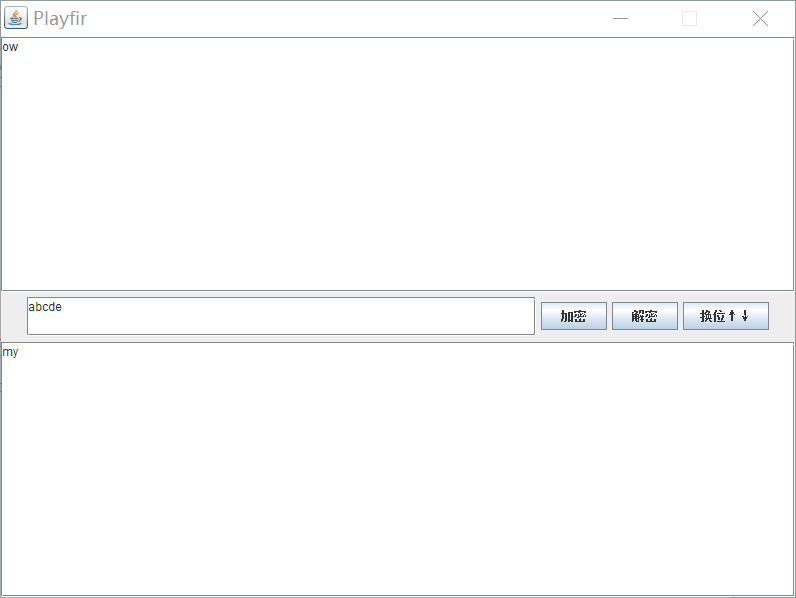
**解密** 密文：2072244336 密钥：12345678912345 -->明文：1520149488

**Playfir**

**加密** 明文：my 密钥：abcde -->密文：ow



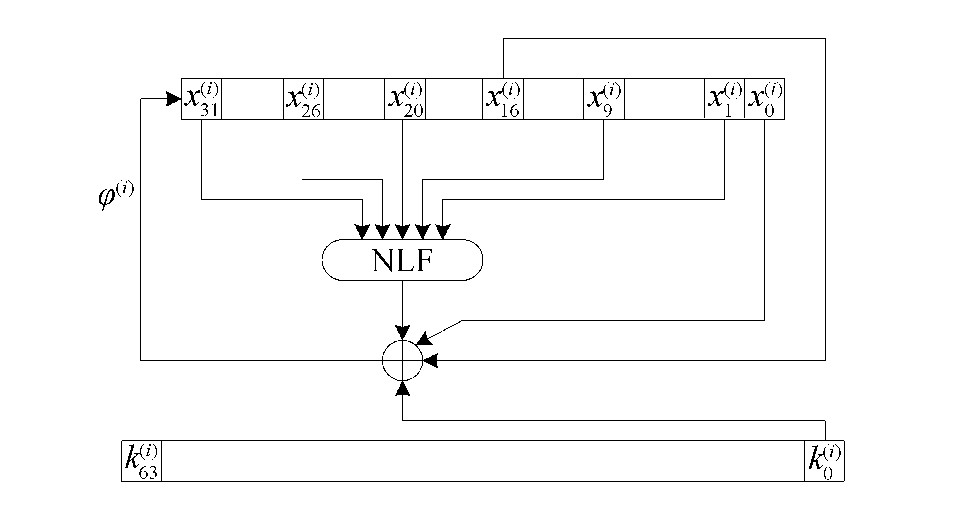
**解密** 密文：ow 密钥：abcde -->明文：my



**keeloq算法的核心思想**

用8byte密钥加密4byte明文，从而得到4byte密文，解密同理。KeeLoq算法过程中需要定义一个数据寄存器，用于存放4byte明文或密文y31-y0，以及一个密钥寄存器，用于存放8byte密钥k63-k0。

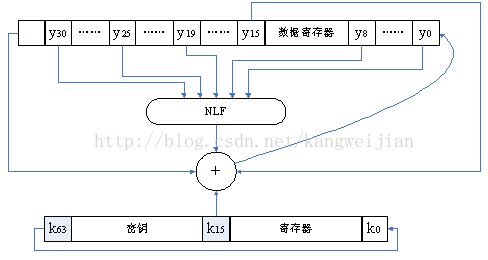
**加密**



首先定义一个非线性表，有5bits输入码，1bit输出码。它在数据寄存器中间隔均匀地取5bits: y31、y26、y20、y9、y1，通过NLF运算产生一个输出码。输出码再与数据寄存器中的y16与y0以及密钥寄存器中的k0进行异或运算后生成1bit加密数据码。每生成1bit加密数据码，密钥寄存器和数据寄存器分别进行移位，密钥寄存器作循环移位，加密数据码作为数据寄存器移位的输入，重复上述步骤528次后，得到4byte的输出密文。

NLF(x4,x3,x2,x1,x0)=x4x3x2^x4x3x1^x4x2x0^x4x1x0^x4x2^x4x0^x3x2^x3x0^x2x1^x1x0^x1^x0

**解密**



**解密流程图**

与加密基本一致，只是其中运算数据数据位发生变化。非线性表的5bits输入码改成从数据寄存器中间隔均匀地取固定5bits:y30、y25、y19、y8、y0。产生1bit输出码后输出码再与数据寄存器中的y31与y15以及密钥寄存器中的k15进行异或运算后生成1bit解密数据码。每输出1bit解密数据码后，密钥寄存器和数据寄存器分别进行移位，密钥寄存器作循环移位，解密数据码作为数据寄存器移位的输入，重复上述步骤528次后，还原出4byte的明文。

**KeeLoq特点**

KeeLoq算法的NLF运算规则，使得一个很小的输入变化量，也会造成很大的输出变化量，就无法通过输入微小的变化来观察分析输出的变化，从而破解出密钥，使得KeeLoq算法具有安全性高特点。

**源代码**











