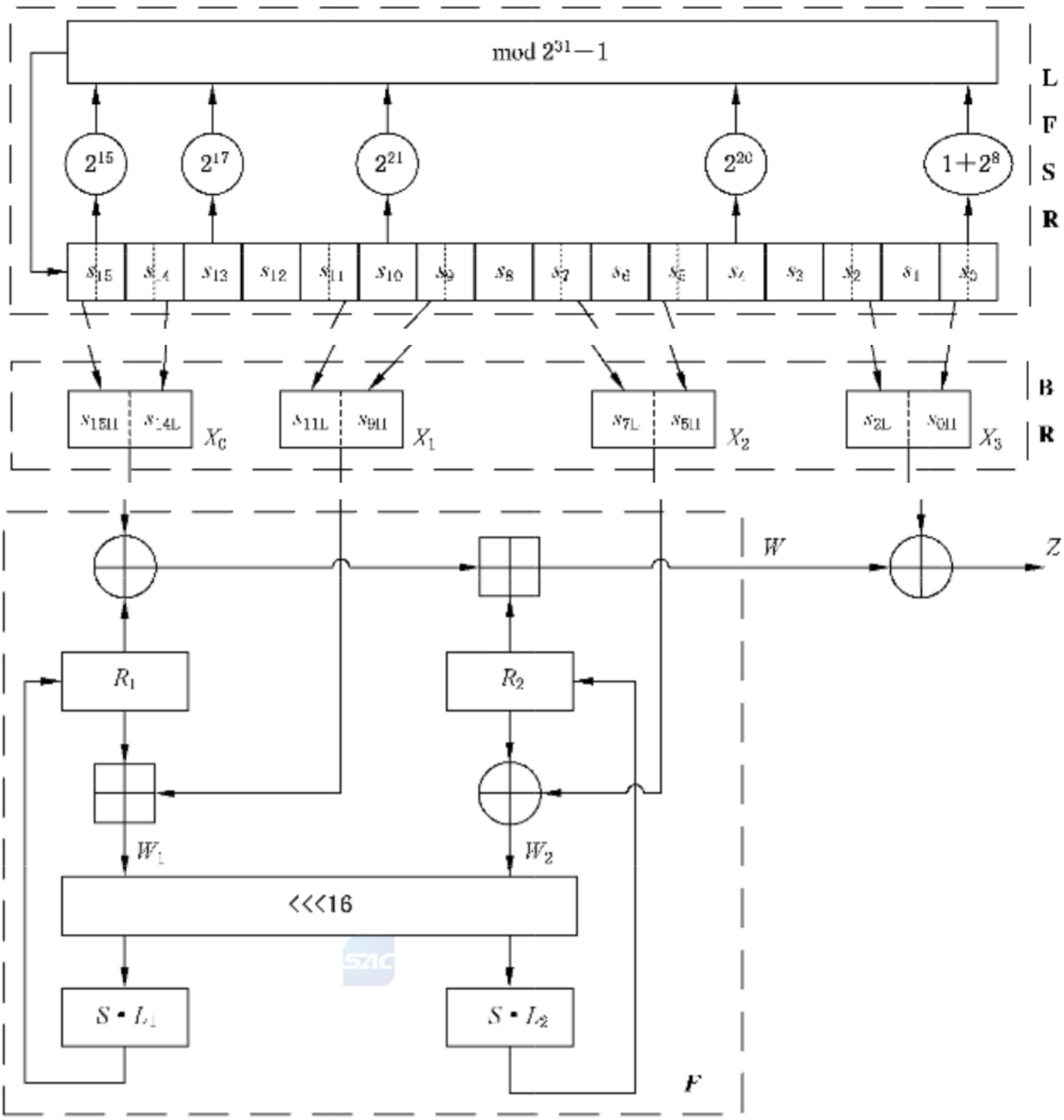
ZUC的C/C++实现-20307130135李钧

根据GMT0001.1-2012(祖冲之序列密码算法第1部分-算法描述)以及第2、第3部分说明，祖冲之算法结构图如下所示，由线性反馈移位寄存器LFSR、比特重组BR和非线性函数F组成



利用C/C++实现祖冲之序列密码算法，以及基于此的机密性算法、完整性算法，文件结构如下

```
E:\0Tree_Down\crypto\0last2\ZUC>tree /F
卷 新加卷 的文件夹 PATH 列表
卷序列号为 2A31-0171
E:.
│  CMakeLists.txt
│  Zmain.cpp
│  ZUC.cpp
│  ZUC.exe
│  ZUC_H.h
│  ZUC_jm.cpp
│  ZUC_wz.cpp
│
└─build
    │  CMakeCache.txt
    │  cmake_install.cmake
    │  compile_commands.json
    │  Makefile
    │
    ├─cmake
```

祖冲之算法描述

常量、函数的声明与定义

在头文件中定义两个 32 比特的 S 盒、定义用于算法初始化的字符串常量 D，同时声明一些必要的符号、函数，如下所示，具体定义参考文档内容放置在 cpp 源文件中。

```
//一些必要符号
typedef unsigned char uint8;
typedef unsigned int uint32;
extern uint32 LFSR[16];//线性反馈以为寄存器
extern uint32 X[4];//比特重组输出的 4 个 32 比特
extern uint32 R1, R2;//非线性函数 F 的 2 个 32 比特记忆单元变量
extern uint32 W;//非线性函数 F 输出的 32 比特字
//必要的函数声明
uint32 Rmv(uint32 x, int move);// 循环移位
uint32 mod_add(uint32 a, uint32 b);// 模 2^31-1 加法
uint32 mod_2exp_mul(uint32 x, int exp);// 模 2^31-1 乘法
uint32 L1(uint32 x);//两个线性变换
uint32 L2(uint32 x);
void LFSRWithInitMode(uint32 u);//初始化模式
void LFSRWithWorkMode();//工作模式（与初始化基本一样）
void BitReconstruction();//比特重组 BR，十六位为 HL 分界
uint32 S(uint32 a);//S 盒输入输出，附录 A
void F();//非线性函数 F
```

```
void Key_Init(uint8 *k, uint8 *iv);//密钥装入，初始化阶段
uint32 *KeyStream_Generator(int keylen);//工作阶段，最后输出一个 32 比特的密钥字 Z
```

两个具体函数 Key_Init()和 KeyStream_Generator()定义如下

```
/密钥装入，初始化阶段
void Key_Init(uint8 *k, uint8 *iv){
    for(int i = 0; i < 16; i++)
        {LFSR[i] = (k[i] << 23) | (D[i] << 8) | iv[i];}
    R1 = R2 = 0;
    for(int i = 0; i < 32; i++){
        BitReconstruction();
        F();
        LFSRWithInitMode(W >> 1);
    }
}
//工作阶段，最后输出一个 32 比特的密钥字 Z
uint32 *KeyStream_Generator(int keylen){
    uint32 Z = 0, i = 0;
    uint32 *keystream = (uint32 *)malloc((keylen + 1) * sizeof(uint32));
    BitReconstruction();
    F();
    LFSRWithWorkMode();
    for (i = 0; i < keylen; i++){
        BitReconstruction();
        F();
        keystream[i] = W ^ X[3];
        LFSRWithWorkMode();
    }
    return keystream;
}
```

算法运行

祖冲之算法的输入参数为初始密钥 k、初始向量 iv 和正整数 L，输出参数为 L 个密钥字 Z。然后按照上述定义的初始化方法将初始密钥 k 和初始向量 iv 装入到 LFSR 的寄存器单元变量 s0~s15 中，作为 LFSR 的初态等等。工作步骤也如上述密钥生成算法一样。

其他相关基础函数的定义具体见源代码。

基于 ZUC 的机密性算法

在头文件中声明相关输入参数、函数如下所示。

```
int get_L_jm(uint32 length);
extern uint8 KEY[16];
extern uint8 IV[16];
extern uint8 CK[16];//机密性密钥
extern uint8 COUNT[4];//计数器
extern uint8 BEARER;//承载层表识
extern uint8 DIRECTION;//传输方向标识
extern uint32 LENGTH;//明文消息流的比特长度
extern uint32 IBS[];//LENGTH
extern uint32 OBS[];//LENGTH
void ZUC_jm_Init(uint8 *ck, uint8 *count,
                uint8 *KEY, uint8 *IV,
                uint8 bearer, uint8 direction, int &keylen);//基于祖冲之算法的机密性算法初始化
void ZUC_jm_en(uint32 *ibs, uint32 *k, uint32 length, uint32 *obs, int keylen);//加密生成OBS 输出比特流
```

由于本算法基于 ZUC 算法，对密钥 k、初始向量 iv 的初始化与上述密钥装入不同，故在此将密钥初始化的接口定义为 ZUC_jm_Init；而且第三部分的输出与此部分的输出不同，故将此部分机密性算法的输出接口定义为 ZUC_jm_en，具体定义参考官方文档描述如下：

```
int get_L_jm(uint32 length){
    uint32 t;
```

```
    t = length % 32;
    if(t == 0){return length/32;}
    else{return length/32 + 1;}
}
void ZUC_jm_Init(uint8 *ck, uint8 *count,
                uint8 *KEY, uint8 *IV,
                uint8 bearer, uint8 direction, int &keylen){
    //init KEY
    keylen = get_L_jm(LENGTH);
    for(uint8 i = 0; i < 16; i++){KEY[i] = ck[i];}
    //init IV
    for(uint8 i = 0; i < 4; i++) IV[i] = count[i];
    IV[4] = ((bearer << 3) & 0xf8) | ((direction << 2) & 0x4);
    IV[5] = 0x00; IV[6] = 0x00; IV[7] = 0x00;
    for(uint8 i = 8; i < 16; i++) IV[i] = IV[i - 8];
}


void ZUC_jm_en(uint32 *ibs, uint32 *k, uint32 length, uint32 *obs, int keylen){
    for(int i = 0; i < length/32; i++){obs[i] = (ibs[i] ^ k[i]);}
    printf("OBS is:\n");
    for(int i = 0; i < keylen; i++) printf("%08x ", OBS[i]);
}
```

基于 ZUC 的完整性算法

为了与第二部分的机密性算法区分开来，在头文件中重新声明了相关参数、函数

```
extern uint8 IK[16];
extern uint8 COUNT_w[4];//计数器
extern uint8 BEARER_w;//承载层表识
extern uint8 DIRECTION_w;//传输方向标识
extern uint32 LENGTH_w;//明文消息流的比特长度
extern uint32 M[];
extern uint32 MAC;
```

```
void ZUC_wz_Init(uint8 *ik, uint8 *count,
                uint8 *KEY, uint8 *IV,
                uint8 bearer, uint8 direction, int &keylen);//基于祖冲之算法的完整性算法初始化
uint32 ZUC_MAC(uint32 *m, uint32 *k, uint32 length);//生成 32 位比特的哈希值
uint32 GET_WORD(uint32 *DATA, uint32 i);//获取第 i 位往后的 32bit 字
uint8 GET_BIT(uint32 *DATA, uint32 i);//获取数据流中第 i 位的 bit
int get_L_wz(uint32 length);//根据输入数据流的长度获取密钥长度 L
```

本算法的初始化、输出接口与上述不同，重新定义接口如下所示

```
void ZUC_wz_Init(uint8 *ik, uint8 *count,
                uint8 *KEY, uint8 *IV,
                uint8 bearer, uint8 direction, int &keylen){
    keylen = get_L_wz(LENGTH_w);
    //init KEY
    for(uint8 i = 0; i < 16; i++){KEY[i] = ik[i];}
    //init IV
    for(uint8 i = 0; i < 4; i++) IV[i] = count[i];
    IV[4] = (bearer << 3) & 0xf8;
    IV[5] = 0x00; IV[6] = 0x00; IV[7] = 0x00;
    IV[8] = IV[0] ^ ((direction & 1) << 7);
    for(uint8 i = 9; i < 14; i++) IV[i] = IV[i - 8];
    IV[14] = IV[6] ^ ((direction & 1) << 7);
    IV[15] = IV[7];
}
uint32 ZUC_MAC(uint32 *m, uint32 *k, uint32 length){
    uint32 T = 0, mac;
    for(uint32 i = 0; i < length; i++){if(GET_BIT(m, i)){T = T ^ GET_WORD(k, i);}}
    T = T ^ GET_WORD(k, length);
    int L = get_L_wz(length);
    mac = T ^ k[L-1];
    printf("\nMAC is %08x\n", mac);
    return mac;
```

```
}
```

结果测试

机密性算法的输出结果展示

CK	—17 3d 14 ba 50 03 73 1d 7a 60 04 94 70 f0 0a 29
COUNT	—66035492
BEARER	—f
DIRECTION	—0
LENGTH	—c1

IBS：

6cf65340 735552ab 0c9752fa 6f9025fe 0bd675d9 005875b2 00000000

OBS：

a6c85fc6 6afb8533 aafc2518 dfe78494 0ee1e4b0 30238cc8 00000000

```
PS E:\0Tree_Down\crypto\0last2\ZUC> ."E:/0Tree_Down/crypto/0last2/ZUC/ZUC.exe"
17 3d 14 ba 50 03 73 1d 7a 60 04 94 70 f0 0a 29
KEY init above, and count below
66035492
this is IV below:
66 03 54 92 78 00 00 00
66 03 54 92 78 00 00 00


this is keylen: 7

this is the 0 key: ca3e0c86
this is the 1 key: 19aed798
this is the 2 key: a66b77e2
this is the 3 key: b077a16a
this is the 4 key: 05379169
this is the 5 key: 307bf97a
this is the 6 key: 104b5a43
this is the 7 key: 00000000


this is the OBS of ZUC_jm:
a6c85fc6 1e9b8533 aafc2518 dfe78494 0ee1e4b0 30238cc8 00000000
```

完整性算法的输出结果展示：

第一组实例：

IK —00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

COUNT —0

BEARER —0

DIRECTION —0

LENGTH —1

M:00000000

MAC:c8a9595e

```
PS E:\0Tree_Down\crypto\0last2\ZUC> ."E:/0Tree_Down/crypto/0last2/ZUC/ZUC.exe"
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
KEY init above, and count below
00000000this is IV below:
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
BEARER is 0, DIRECTION is 0

this is keylen: 7

this is the 0 key: 27bede74
this is the 1 key: 018082da
this is the 2 key: 87d4e5b6
this is the 3 key: 9f18bf66
this is the 4 key: 32070e0f
this is the 5 key: 39b7b692
this is the 6 key: b4673edc
this is the 7 key: 00000000


MAC is c8a9595e
```

第二组实例：

IK —c9 e6 ce c4 60 7c 72 db 00 0a ef a8 83 85 ab 0a

COUNT —a94059da

BEARER —a

DIRECTION —1

LENGTH —241

M:

983b41d4 7d780c9e 1ad11d7e b70391b1 de0b35da 2dc62f83 e7b78d63 06ca0ea0 7e941b7b
e91348f9 fcb170e2 217fecd9 7f9f68ad b16e5d7d 21e569d2 80ed775c ebde3f40 93c53881
00000000

MAC:fae8ff0b

```
PS E:\0Tree_Down\crypto\0last2\ZUC> ."E:/0Tree_Down/crypto/0last2/ZUC/ZUC.exe"
c9 e6 ce c4 60 7c 72 db 00 0a ef a8 83 85 ab 0a
KEY init above, and count below
a94059da
this is IV below:
a9 40 59 da 50 00 00 00
29 40 59 da 50 00 80 00
BEARER is a, DIRECTION is 1

this is keylen: 21

into the ZUC_MAC, length: 577

MAC is fae8ff0b
```