Приложение А

(обязательное)

Листинг программы микроконтроллера на языке С

```
//
//file belt.h
//

/*
*****************************************************************************
******
Шифрование в режиме счетчика
*****************************************************************************
******
*/

#define B_PER_W 16
#define O_PER_W (B_PER_W / 8)
typedef unsigned short WORD;
typedef unsigned long u32;
typedef signed long i32;
typedef unsigned char u8;
typedef signed char i8;
typedef u8 octet;
typedef unsigned int size_t;

typedef struct
{
 u32 key[8];             //форматированный ключ
 u32 ctr[4];             //счетчик
 octet block[16];  //блок гаммы
 size_t reserved;  //резерв октетов гаммы
} belt_ctr_st;

/*
*****************************************************************************
******
Ускорители

Реализованы быстрые операции над блоками и полублоками belt. Блок
представляется либо как [16]octet, либо как [4]u32,
либо как [W_OF_B(128)]word.

Суффикс U32 в именах макросов и функций означает, что данные
интерпретируются
как массив u32. Суффикс W означает, что данные интерпретируются как
массив word.
*****************************************************************************
******
*/

#define beltBlockIncU32(block)\
 if ((((u32*)(block))[0] += 1) == 0 &&\
       (((u32*)(block))[1] += 1) == 0 &&\
       (((u32*)(block))[2] += 1) == 0)\
       ((u32*)(block))[3] += 1\

#define beltBlockCopy(dest, src)\
 ((WORD*)(dest))[0] = ((const WORD*)(src))[0],\
 ((WORD*)(dest))[1] = ((const WORD*)(src))[1],\
 ((WORD*)(dest))[2] = ((const WORD*)(src))[2],\
```

```
 ((WORD*)(dest))[3] = ((const WORD*)(src))[3]\

#define beltBlockXor2(dest, src)\
 ((WORD*)(dest))[0] ^= ((const WORD*)(src))[0],\
 ((WORD*)(dest))[1] ^= ((const WORD*)(src))[1],\
 ((WORD*)(dest))[2] ^= ((const WORD*)(src))[2],\
 ((WORD*)(dest))[3] ^= ((const WORD*)(src))[3]\

/*!
*****************************************************************************
******
Блоб -- объект в памяти определенного размера. В функциях работы с
блобами
используются их дескрипторы -- "умные" указатели. С дескрипторами можно
работать как с обычными указателями, т.е. использовать их в функциях типа
memcpy, memset. Дополнительно по указателю можно определить размер блоба.

Реализация работы с блобами может быть платформенно-зависимой.

Реализация должна гарантировать защиту содержимого блобов от утечек,
например, через файл подкачки. Поэтому в блобах рекомендуется размещать
ключи и другие критические объекты.

В функциях работы с блобами дескрипторы входных блобов корректны.
*****************************************************************************
******
*/

// память для блобов выделяется страницами
#define BLOB_PAGE_SIZE 1024

// требуется страниц
#define blobPageCount(size)\
 (((size) + sizeof(size_t) + BLOB_PAGE_SIZE - 1) / BLOB_PAGE_SIZE)

// требуется памяти на страницах
#define blobActualSize(size)\
 (blobPageCount(size) * BLOB_PAGE_SIZE)

// heap-указатель для блоба
#define blobPtrOf(blob) ((size_t*)blob - 1)

// размер блоба
#define blobSizeOf(blob) (*blobPtrOf(blob))

// страничный размер блоба
#define blobActualSizeOf(blob) (blobActualSize(blobSizeOf(blob)))

// блоб для heap-указателя
#define blobValueOf(ptr) ((blob_t)((size_t*)ptr + 1))

// дескриптор блоба
typedef void* blob_t;

/*     Инициализация шифрования в режиме CTR

 По ключу [len]key и синхропосылке iv в state формируются
 структуры данных, необходимые для шифрования в режиме CTR.
 len == 16 || len == 24 || len == 32.
 По адресу state зарезервировано beltCTR_keep() октетов.
 Буферы key и state могут пересекаться.
*/
void beltCTRStart(
```

```
 void* state,                  //[out] состояние
 const octet key[],            //[in] ключ
 size_t len,                   //[in] длина ключа в октетах
 const octet iv[16]            //[in] синхропосылка
);

/*     Зашифрование фрагмента в режиме CTR

 Буфер [count]buf зашифровывается в режиме CTR на ключе, размещенном
 в state.
 beltCTRStart() < beltCTRStepE()*.
*/
void beltCTRStepE(
 void* buf,              //[in/out] открытый текст / шифртекст
 size_t count,           //[in] число октетов текста
 void* state             //[in/out] состояние
);

/*     Расшифрование фрагмента в режиме CTR
 Зашифрование в режиме CTR не отличается от расшифрования.
*/
#define beltCTRStepD beltCTRStepE

/*     Шифрование в режиме CTR

 Буфер [count]src зашифровывается или расшифровывается на ключе
 [len]key с использованием синхропосылки iv. Результат шифрования
 размещается в буфере [count]dest.
 {ERR_BAD_INPUT} len == 16 || len == 24 || len == 32.
 ERR_OK, если шифрование завершено успешно, и код ошибки
 в противном случае.
 Буферы могут пересекаться.
*/

//
//file belt.cpp
//

/*
*******************************************************************************
******
STB 34.101.31 (belt): CTR encryption
*******************************************************************************
******
*/

#include "belt.h"
#include <string.h>
#include <avr/pgmspace.h>

/*
*******************************************************************************
******
Загрузка
*******************************************************************************
******
*/

void u32From(u32 dest[], const void* src, size_t count)
{
 memmove(dest, src, count);
 if (count % 4)
       memset((octet*)dest + count, 0, 4 - count % 4);
```

```
	}

	/*
	*****************************************************************************
******
	Расширение ключа
	*****************************************************************************
******
	*/

	void beltKeyExpand(u32 key_[8], const octet key[], size_t len)
	{
	 u32From(key_, key, len);
	 if (len == 16)
	 {
		key_[4] = key_[0];
		key_[5] = key_[1];
		key_[6] = key_[2];
		key_[7] = key_[3];
	 }
	 else if (len == 24)
	 {
		key_[6] = key_[0] ^ key_[1] ^ key_[2];
		key_[7] = key_[3] ^ key_[4] ^ key_[5];
	 }
	}

	/*
	*****************************************************************************
******
	H-блок
	*****************************************************************************
******
	*/
	/*
	static const octet H[256] = {
	 0xB1,0x94,0xBA,0xC8,0x0A,0x08,0xF5,0x3B,0x36,0x6D,0x00,0x8E,0x58,0x4A,0x
5D,0xE4,
	 0x85,0x04,0xFA,0x9D,0x1B,0xB6,0xC7,0xAC,0x25,0x2E,0x72,0xC2,0x02,0xFD,0x
CE,0x0D,
	 0x5B,0xE3,0xD6,0x12,0x17,0xB9,0x61,0x81,0xFE,0x67,0x86,0xAD,0x71,0x6B,0x
89,0x0B,
	 0x5C,0xB0,0xC0,0xFF,0x33,0xC3,0x56,0xB8,0x35,0xC4,0x05,0xAE,0xD8,0xE0,0x
7F,0x99,
	 0xE1,0x2B,0xDC,0x1A,0xE2,0x82,0x57,0xEC,0x70,0x3F,0xCC,0xF0,0x95,0xEE,0x
8D,0xF1,
	 0xC1,0xAB,0x76,0x38,0x9F,0xE6,0x78,0xCA,0xF7,0xC6,0xF8,0x60,0xD5,0xBB,0x
9C,0x4F,
	 0xF3,0x3C,0x65,0x7B,0x63,0x7C,0x30,0x6A,0xDD,0x4E,0xA7,0x79,0x9E,0xB2,0x
3D,0x31,
	 0x3E,0x98,0xB5,0x6E,0x27,0xD3,0xBC,0xCF,0x59,0x1E,0x18,0x1F,0x4C,0x5A,0x
B7,0x93,
	 0xE9,0xDE,0xE7,0x2C,0x8F,0x0C,0x0F,0xA6,0x2D,0xDB,0x49,0xF4,0x6F,0x73,0x
96,0x47,
	 0x06,0x07,0x53,0x16,0xED,0x24,0x7A,0x37,0x39,0xCB,0xA3,0x83,0x03,0xA9,0x
8B,0xF6,
	 0x92,0xBD,0x9B,0x1C,0xE5,0xD1,0x41,0x01,0x54,0x45,0xFB,0xC9,0x5E,0x4D,0x
0E,0xF2,
	 0x68,0x20,0x80,0xAA,0x22,0x7D,0x64,0x2F,0x26,0x87,0xF9,0x34,0x90,0x40,0x
55,0x11,
	 0xBE,0x32,0x97,0x13,0x43,0xFC,0x9A,0x48,0xA0,0x2A,0x88,0x5F,0x19,0x4B,0x
09,0xA1,
```

```
      0x7E,0xCD,0xA4,0xD0,0x15,0x44,0xAF,0x8C,0xA5,0x84,0x50,0xBF,0x66,0xD2,0x
E8,0x8A,
      0xA2,0xD7,0x46,0x52,0x42,0xA8,0xDF,0xB3,0x69,0x74,0xC5,0x51,0xEB,0x23,0x
29,0x21,
      0xD4,0xEF,0xD9,0xB4,0x3A,0x62,0x28,0x75,0x91,0x14,0x10,0xEA,0x77,0x6C,0x
DA,0x1D,
     };

     const octet* beltH()
     {
      return H;
     }
     */

     /*
     ************************************************************************
******
     Расширенные H-блоки

      Описание построено с помощью функции:

      void beltExtendBoxes()
      {
            unsigned r, x;
            u32 y;
            for (r = 5; r < 32; r += 8)
            {
                  printf("static const u32 H%u[256] = {", r);
                  for (x = 0; x < 256; x++)
                        y = H[x],
                        y = y << r | y >> (32 - r),
                        printf(x % 8 ? "0x%08X," : "\n\t0x%08X,", y);
                  printf("\n};\n");
            }
      }
     ************************************************************************
******
     */

     static const u32 H5[256] PROGMEM = {
      0x00001620,0x00001280,0x00001740,0x00001900,0x00000140,0x00000100,0x0000
1EA0,0x00000760,
      0x000006C0,0x00000DA0,0x00000000,0x000011C0,0x00000B00,0x00000940,0x0000
0BA0,0x00001C80,
      0x000010A0,0x00000080,0x00001F40,0x000013A0,0x00000360,0x000016C0,0x0000
18E0,0x00001580,
      0x000004A0,0x000005C0,0x00000E40,0x00001840,0x00000040,0x00001FA0,0x0000
19C0,0x000001A0,
      0x00000B60,0x00001C60,0x00001AC0,0x00000240,0x000002E0,0x00001720,0x0000
0C20,0x00001020,
      0x00001FC0,0x00000CE0,0x000010C0,0x000015A0,0x00000E20,0x00000D60,0x0000
1120,0x00000160,
      0x00000B80,0x00001600,0x00001800,0x00001FE0,0x00000660,0x00001860,0x0000
0AC0,0x00001700,
      0x000006A0,0x00001880,0x000000A0,0x000015C0,0x00001B00,0x00001C00,0x0000
0FE0,0x00001320,
      0x00001C20,0x00000560,0x00001B80,0x00000340,0x00001C40,0x00001040,0x0000
0AE0,0x00001D80,
      0x00000E00,0x000007E0,0x00001980,0x00001E00,0x000012A0,0x00001DC0,0x0000
11A0,0x00001E20,
      0x00001820,0x00001560,0x00000EC0,0x00000700,0x000013E0,0x00001CC0,0x0000
0F00,0x00001940,
```

```
      0x00001EE0,0x000018C0,0x00001F00,0x00000C00,0x00001AA0,0x00001760,0x0000
1380,0x000009E0,
      0x00001E60,0x00000780,0x00000CA0,0x00000F60,0x00000C60,0x00000F80,0x0000
0600,0x00000D40,
      0x00001BA0,0x000009C0,0x000014E0,0x00000F20,0x000013C0,0x00001640,0x0000
07A0,0x00000620,
      0x000007C0,0x00001300,0x000016A0,0x00000DC0,0x000004E0,0x00001A60,0x0000
1780,0x000019E0,
      0x00000B20,0x000003C0,0x00000300,0x000003E0,0x00000980,0x00000B40,0x0000
16E0,0x00001260,
      0x00001D20,0x00001BC0,0x00001CE0,0x00000580,0x000011E0,0x00000180,0x0000
01E0,0x000014C0,
      0x000005A0,0x00001B60,0x00000920,0x00001E80,0x00000DE0,0x00000E60,0x0000
12C0,0x000008E0,
      0x000000C0,0x000000E0,0x00000A60,0x000002C0,0x00001DA0,0x00000480,0x0000
0F40,0x000006E0,
      0x00000720,0x00001960,0x00001460,0x00001060,0x00000060,0x00001520,0x0000
1160,0x00001EC0,
      0x00001240,0x000017A0,0x00001360,0x00000380,0x00001CA0,0x00001A20,0x0000
0820,0x00000020,
      0x00000A80,0x000008A0,0x00001F60,0x00001920,0x00000BC0,0x000009A0,0x0000
01C0,0x00001E40,
      0x00000D00,0x00000400,0x00001000,0x00001540,0x00000440,0x00000FA0,0x0000
0C80,0x000005E0,
      0x000004C0,0x000010E0,0x00001F20,0x00000680,0x00001200,0x00000800,0x0000
0AA0,0x00000220,
      0x000017C0,0x00000640,0x000012E0,0x00000260,0x00000860,0x00001F80,0x0000
1340,0x00000900,
      0x00001400,0x00000540,0x00001100,0x00000BE0,0x00000320,0x00000960,0x0000
0120,0x00001420,
      0x00000FC0,0x000019A0,0x00001480,0x00001A00,0x000002A0,0x00000880,0x0000
15E0,0x00001180,
      0x000014A0,0x00001080,0x00000A00,0x000017E0,0x00000CC0,0x00001A40,0x0000
1D00,0x00001140,
      0x00001440,0x00001AE0,0x000008C0,0x00000A40,0x00000840,0x00001500,0x0000
1BE0,0x00001660,
      0x00000D20,0x00000E80,0x000018A0,0x00000A20,0x00001D60,0x00000460,0x0000
0520,0x00000420,
      0x00001A80,0x00001DE0,0x00001B20,0x00001680,0x00000740,0x00000C40,0x0000
0500,0x00000EA0,
      0x00001220,0x00000280,0x00000200,0x00001D40,0x00000EE0,0x00000D80,0x0000
1B40,0x000003A0,
     };
     static const u32 H13[256] PROGMEM = {
      0x00162000,0x00128000,0x00174000,0x00190000,0x00014000,0x00010000,0x001E
A000,0x00076000,
      0x0006C000,0x000DA000,0x00000000,0x0011C000,0x000B0000,0x00094000,0x000B
A000,0x001C8000,
      0x0010A000,0x00008000,0x001F4000,0x0013A000,0x00036000,0x0016C000,0x0018
E000,0x00158000,
      0x0004A000,0x0005C000,0x000E4000,0x00184000,0x00004000,0x001FA000,0x0019
C000,0x0001A000,
      0x000B6000,0x001C6000,0x001AC000,0x00024000,0x0002E000,0x00172000,0x000C
2000,0x00102000,
      0x001FC000,0x000CE000,0x0010C000,0x0015A000,0x000E2000,0x000D6000,0x0011
2000,0x00016000,
      0x000B8000,0x00160000,0x00180000,0x001FE000,0x00066000,0x00186000,0x000A
C000,0x00170000,
      0x0006A000,0x00188000,0x0000A000,0x0015C000,0x001B0000,0x001C0000,0x000F
E000,0x00132000,
      0x001C2000,0x00056000,0x001B8000,0x00034000,0x001C4000,0x00104000,0x000A
E000,0x001D8000,
```

```
      0x000E0000,0x0007E000,0x00198000,0x001E0000,0x0012A000,0x001DC000,0x0011
A000,0x001E2000,
      0x00182000,0x00156000,0x000EC000,0x00070000,0x0013E000,0x001CC000,0x000F
0000,0x00194000,
      0x001EE000,0x0018C000,0x001F0000,0x000C0000,0x001AA000,0x00176000,0x0013
8000,0x0009E000,
      0x001E6000,0x00078000,0x000CA000,0x000F6000,0x000C6000,0x000F8000,0x0006
0000,0x000D4000,
      0x001BA000,0x0009C000,0x0014E000,0x000F2000,0x0013C000,0x00164000,0x0007
A000,0x00062000,
      0x0007C000,0x00130000,0x0016A000,0x000DC000,0x0004E000,0x001A6000,0x0017
8000,0x0019E000,
      0x000B2000,0x0003C000,0x00030000,0x0003E000,0x00098000,0x000B4000,0x0016
E000,0x00126000,
      0x001D2000,0x001BC000,0x001CE000,0x00058000,0x0011E000,0x00018000,0x0001
E000,0x0014C000,
      0x0005A000,0x001B6000,0x00092000,0x001E8000,0x000DE000,0x000E6000,0x0012
C000,0x0008E000,
      0x0000C000,0x0000E000,0x000A6000,0x0002C000,0x001DA000,0x00048000,0x000F
4000,0x0006E000,
      0x00072000,0x00196000,0x00146000,0x00106000,0x00006000,0x00152000,0x0011
6000,0x001EC000,
      0x00124000,0x0017A000,0x00136000,0x00038000,0x001CA000,0x001A2000,0x0008
2000,0x00002000,
      0x000A8000,0x0008A000,0x001F6000,0x00192000,0x000BC000,0x0009A000,0x0001
C000,0x001E4000,
      0x000D0000,0x00040000,0x00100000,0x00154000,0x00044000,0x000FA000,0x000C
8000,0x0005E000,
      0x0004C000,0x0010E000,0x001F2000,0x00068000,0x00120000,0x00080000,0x000A
A000,0x00022000,
      0x0017C000,0x00064000,0x0012E000,0x00026000,0x00086000,0x001F8000,0x0013
4000,0x00090000,
      0x00140000,0x00054000,0x00110000,0x000BE000,0x00032000,0x00096000,0x0001
2000,0x00142000,
      0x000FC000,0x0019A000,0x00148000,0x001A0000,0x0002A000,0x00088000,0x0015
E000,0x00118000,
      0x0014A000,0x00108000,0x000A0000,0x0017E000,0x000CC000,0x001A4000,0x001D
0000,0x00114000,
      0x00144000,0x001AE000,0x0008C000,0x000A4000,0x00084000,0x00150000,0x001B
E000,0x00166000,
      0x000D2000,0x000E8000,0x0018A000,0x000A2000,0x001D6000,0x00046000,0x0005
2000,0x00042000,
      0x001A8000,0x001DE000,0x001B2000,0x00168000,0x00074000,0x000C4000,0x0005
0000,0x000EA000,
      0x00122000,0x00028000,0x00020000,0x001D4000,0x000EE000,0x000D8000,0x001B
4000,0x0003A000,
     };
     static const u32 H21[256] PROGMEM = {
      0x16200000,0x12800000,0x17400000,0x19000000,0x01400000,0x01000000,0x1EA0
0000,0x07600000,
      0x06C00000,0x0DA00000,0x00000000,0x11C00000,0x0B000000,0x09400000,0x0BA0
0000,0x1C800000,
      0x10A00000,0x00800000,0x1F400000,0x13A00000,0x03600000,0x16C00000,0x18E0
0000,0x15800000,
      0x04A00000,0x05C00000,0x0E400000,0x18400000,0x00400000,0x1FA00000,0x19C0
0000,0x01A00000,
      0x0B600000,0x1C600000,0x1AC00000,0x02400000,0x02E00000,0x17200000,0x0C20
0000,0x10200000,
      0x1FC00000,0x0CE00000,0x10C00000,0x15A00000,0x0E200000,0x0D600000,0x1120
0000,0x01600000,
      0x0B800000,0x16000000,0x18000000,0x1FE00000,0x06600000,0x18600000,0x0AC0
0000,0x17000000,
```

```
      0x06A00000,0x18800000,0x00A00000,0x15C00000,0x1B000000,0x1C000000,0x0FE0
0000,0x13200000,
      0x1C200000,0x05600000,0x1B800000,0x03400000,0x1C400000,0x10400000,0x0AE0
0000,0x1D800000,
      0x0E000000,0x07E00000,0x19800000,0x1E000000,0x12A00000,0x1DC00000,0x11A0
0000,0x1E200000,
      0x18200000,0x15600000,0x0EC00000,0x07000000,0x13E00000,0x1CC00000,0x0F00
0000,0x19400000,
      0x1EE00000,0x18C00000,0x1F000000,0x0C000000,0x1AA00000,0x17600000,0x1380
0000,0x09E00000,
      0x1E600000,0x07800000,0x0CA00000,0x0F600000,0x0C600000,0x0F800000,0x0600
0000,0x0D400000,
      0x1BA00000,0x09C00000,0x14E00000,0x0F200000,0x13C00000,0x16400000,0x07A0
0000,0x06200000,
      0x07C00000,0x13000000,0x16A00000,0x0DC00000,0x04E00000,0x1A600000,0x1780
0000,0x19E00000,
      0x0B200000,0x03C00000,0x03000000,0x03E00000,0x09800000,0x0B400000,0x16E0
0000,0x12600000,
      0x1D200000,0x1BC00000,0x1CE00000,0x05800000,0x11E00000,0x01800000,0x01E0
0000,0x14C00000,
      0x05A00000,0x1B600000,0x09200000,0x1E800000,0x0DE00000,0x0E600000,0x12C0
0000,0x08E00000,
      0x00C00000,0x00E00000,0x0A600000,0x02C00000,0x1DA00000,0x04800000,0x0F40
0000,0x06E00000,
      0x07200000,0x19600000,0x14600000,0x10600000,0x00600000,0x15200000,0x1160
0000,0x1EC00000,
      0x12400000,0x17A00000,0x13600000,0x03800000,0x1CA00000,0x1A200000,0x0820
0000,0x00200000,
      0x0A800000,0x08A00000,0x1F600000,0x19200000,0x0BC00000,0x09A00000,0x01C0
0000,0x1E400000,
      0x0D000000,0x04000000,0x10000000,0x15400000,0x04400000,0x0FA00000,0x0C80
0000,0x05E00000,
      0x04C00000,0x10E00000,0x1F200000,0x06800000,0x12000000,0x08000000,0x0AA0
0000,0x02200000,
      0x17C00000,0x06400000,0x12E00000,0x02600000,0x08600000,0x1F800000,0x1340
0000,0x09000000,
      0x14000000,0x05400000,0x11000000,0x0BE00000,0x03200000,0x09600000,0x0120
0000,0x14200000,
      0x0FC00000,0x19A00000,0x14800000,0x1A000000,0x02A00000,0x08800000,0x15E0
0000,0x11800000,
      0x14A00000,0x10800000,0x0A000000,0x17E00000,0x0CC00000,0x1A400000,0x1D00
0000,0x11400000,
      0x14400000,0x1AE00000,0x08C00000,0x0A400000,0x08400000,0x15000000,0x1BE0
0000,0x16600000,
      0x0D200000,0x0E800000,0x18A00000,0x0A200000,0x1D600000,0x04600000,0x0520
0000,0x04200000,
      0x1A800000,0x1DE00000,0x1B200000,0x16800000,0x07400000,0x0C400000,0x0500
0000,0x0EA00000,
      0x12200000,0x02800000,0x02000000,0x1D400000,0x0EE00000,0x0D800000,0x1B40
0000,0x03A00000,
     };
     static const u32 H29[256] PROGMEM = {
      0x20000016,0x80000012,0x40000017,0x00000019,0x40000001,0x00000001,0xA000
001E,0x60000007,
      0xC0000006,0xA000000D,0x00000000,0xC0000011,0x0000000B,0x40000009,0xA000
000B,0x8000001C,
      0xA0000010,0x80000000,0x4000001F,0xA0000013,0x60000003,0xC0000016,0xE000
0018,0x80000015,
      0xA0000004,0xC0000005,0x4000000E,0x40000018,0x40000000,0xA000001F,0xC000
0019,0xA0000001,
      0x6000000B,0x6000001C,0xC000001A,0x40000002,0xE0000002,0x20000017,0x2000
000C,0x20000010,
```

```
      0xC000001F,0xE000000C,0xC0000010,0xA0000015,0x2000000E,0x6000000D,0x2000
0011,0x60000001,
      0x8000000B,0x00000016,0x00000018,0xE000001F,0x60000006,0x60000018,0xC000
000A,0x00000017,
      0xA0000006,0x80000018,0xA0000000,0xC0000015,0x0000001B,0x0000001C,0xE000
000F,0x20000013,
      0x2000001C,0x60000005,0x8000001B,0x40000003,0x4000001C,0x40000010,0xE000
000A,0x8000001D,
      0x0000000E,0xE0000007,0x80000019,0x0000001E,0xA0000012,0xC000001D,0xA000
0011,0x2000001E,
      0x20000018,0x60000015,0xC000000E,0x00000007,0xE0000013,0xC000001C,0x0000
000F,0x40000019,
      0xE000001E,0xC0000018,0x0000001F,0x0000000C,0xA000001A,0x60000017,0x8000
0013,0xE0000009,
      0x6000001E,0x80000007,0xA000000C,0x6000000F,0x6000000C,0x8000000F,0x0000
0006,0x4000000D,
      0xA000001B,0xC0000009,0xE0000014,0x2000000F,0xC0000013,0x40000016,0xA000
0007,0x20000006,
      0xC0000007,0x00000013,0xA0000016,0xC000000D,0xE0000004,0x6000001A,0x8000
0017,0xE0000019,
      0x2000000B,0xC0000003,0x00000003,0xE0000003,0x80000009,0x4000000B,0xE000
0016,0x60000012,
      0x2000001D,0xC000001B,0xE000001C,0x80000005,0xE0000011,0x80000001,0xE000
0001,0xC0000014,
      0xA0000005,0x6000001B,0x20000009,0x8000001E,0xE000000D,0x6000000E,0xC000
0012,0xE0000008,
      0xC0000000,0xE0000000,0x6000000A,0xC0000002,0xA000001D,0x80000004,0x4000
000F,0xE0000006,
      0x20000007,0x60000019,0x60000014,0x60000010,0x60000000,0x20000015,0x6000
0011,0xC000001E,
      0x40000012,0xA0000017,0x60000013,0x80000003,0xA000001C,0x2000001A,0x2000
0008,0x20000000,
      0x8000000A,0xA0000008,0x6000001F,0x20000019,0xC000000B,0xA0000009,0xC000
0001,0x4000001E,
      0x0000000D,0x00000004,0x00000010,0x40000015,0x40000004,0xA000000F,0x8000
000C,0xE0000005,
      0xC0000004,0xE0000010,0x2000001F,0x80000006,0x00000012,0x00000008,0xA000
000A,0x20000002,
      0xC0000017,0x40000006,0xE0000012,0x60000002,0x60000008,0x8000001F,0x4000
0013,0x00000009,
      0x00000014,0x40000005,0x00000011,0xE000000B,0x20000003,0x60000009,0x2000
0001,0x20000014,
      0xC000000F,0xA0000019,0x80000014,0x0000001A,0xA0000002,0x80000008,0xE000
0015,0x80000011,
      0xA0000014,0x80000010,0x0000000A,0xE0000017,0xC000000C,0x4000001A,0x0000
001D,0x40000011,
      0x40000014,0xE000001A,0xC0000008,0x4000000A,0x40000008,0x00000015,0xE000
001B,0x60000016,
      0x2000000D,0x8000000E,0xA0000018,0x2000000A,0x6000001D,0x60000004,0x2000
0005,0x20000004,
      0x8000001A,0xE000001D,0x2000001B,0x80000016,0x40000007,0x4000000C,0x0000
0005,0xA000000E,
      0x20000012,0x80000002,0x00000002,0x4000001D,0xE000000E,0x8000000D,0x4000
001B,0xA0000003,
     };

     /*
     ************************************************************************
******
     G-блоки
     ************************************************************************
******
     */
```

```
    #define G5(x)\
     pgm_read_dword_near(H5 + ((x) & 255)) ^ pgm_read_dword_near(H13 + ((x)
>>  8  &  255))  ^  pgm_read_dword_near(H21  +  ((x)  >>  16  &  255))  ^
pgm_read_dword_near(H29 + ((x) >> 24))
    #define G13(x)\
     pgm_read_dword_near(H13 + ((x) & 255)) ^ pgm_read_dword_near(H21 + ((x)
>>  8  &  255))  ^  pgm_read_dword_near(H29  +  ((x)  >>  16  &  255))  ^
pgm_read_dword_near(H5 + ((x) >> 24))
    #define G21(x)\
     pgm_read_dword_near(H21 + ((x) & 255)) ^ pgm_read_dword_near(H29 + ((x)
>>  8  &  255))  ^  pgm_read_dword_near(H5  +  ((x)  >>  16  &  255))  ^
pgm_read_dword_near(H13 + ((x) >> 24))

    /*
    ************************************************************************
******
    Тактовая подстановка

    Макрос R реализует шаги 2.1-2.9 алгоритмов зашифрования и расшифрования.

    На шагах 2.4-2.6 дополнительный регистр e не используется.
    Нужные данные сохраняются в регистрах b и c.

    Параметр-макрос subkey задает порядок использования тактовых ключей:
    порядок subkey = subkey_e используется при зашифровании и расшифровании
    ************************************************************************
******
    */
    #define R(a, b, c, d, K, i, subkey)\
     *b ^= G5(*a + subkey(K, i, 0));\
     *c ^= G21(*d + subkey(K, i, 1));\
     *a -= G13(*b + subkey(K, i, 2));\
     *c += *b;\
     *b += G21(*c + subkey(K, i, 3)) ^ i;\
     *c -= *b;\
     *d += G13(*c + subkey(K, i, 4));\
     *b ^= G21(*a + subkey(K, i, 5));\
     *c ^= G5(*d + subkey(K, i, 6));\

    #define subkey_e(K, i, j) K[(7 * i - 7 + j) % 8]

    /*
    ************************************************************************
******
    Такты зашифрования

    Перестановка содержимого регистров a, b, c, d реализуется перестановкой
    параметров макроса R. После выполнения последнего макроса R и шагов 2.10-
2.12
    алгоритма зашифрования в регистрах a, b, c, d будут находиться значения,
    соответствующие спецификации belt.

    Окончательная перестановка abcd -> bdac реализуется инверсиями:
    a <-> b, c <-> d, b <-> c.
    ************************************************************************
******
    */
    #define E(a, b, c, d, K)\
     R(a, b, c, d, K, 1, subkey_e);\
     R(b, d, a, c, K, 2, subkey_e);\
     R(d, c, b, a, K, 3, subkey_e);\
     R(c, a, d, b, K, 4, subkey_e);\
     R(a, b, c, d, K, 5, subkey_e);\
```

```
	R(b, d, a, c, K, 6, subkey_e);\
	R(d, c, b, a, K, 7, subkey_e);\
	R(c, a, d, b, K, 8, subkey_e);\
	*a ^= *b, *b ^= *a, *a ^= *b;\
	*c ^= *d, *d ^= *c, *c ^= *d;\
	*b ^= *c, *c ^= *b, *b ^= *c;\

	/*
	*****************************************************************************
******
	Зашифрование блока
	*****************************************************************************
******
	*/

	void beltBlockEncr(u32 block[4], const u32 key[8])
	{
	 E((block + 0), (block + 1), (block + 2), (block + 3), key);
	}

	/*
	*****************************************************************************
******
	Расшифрование блока
	*****************************************************************************
******
	*/

	void memXor2(void* dest, const void* src, size_t count)
	{
	 for (; count >= O_PER_W; count -= O_PER_W)
	 {
		*(WORD*)dest ^= *(const WORD*)src;
		src = (const WORD*)src + 1;
		dest = (WORD*)dest + 1;
	 }
	 while (count--)
	 {
		*(octet*)dest ^= *(const octet*)src;
		src = (const octet*)src + 1;
		dest = (octet*)dest + 1;
	 }
	}

	/*
	*****************************************************************************
******
	Шифрование в режиме CTR

	Для ускорения работы счетчик ctr хранится в виде [4]u32. Это позволяет
	зашифровывать счетчик с помощью функции beltBlockEncr(), в которой
	не используется реверс октетов  даже на платформах BIG_ENDIAN.
	Реверс применяется только перед использованием зашифрованного счетчика
	в качестве гаммы.
	*****************************************************************************
******
	*/

	void beltCTRStart(void* state, const octet key[], size_t len,
	 const octet iv[16])
	{
	 belt_ctr_st* st = (belt_ctr_st*)state;
	 beltKeyExpand(st->key, key, len);
```

```
 u32From(st->ctr, iv, 16);
 beltBlockEncr(st->ctr, st->key);
 st->reserved = 0;
}

void beltCTRStepE(void* buf, size_t count, void* state)
{
 belt_ctr_st* st = (belt_ctr_st*)state;
 // есть резерв гаммы?
 if (st->reserved)
 {
       if (st->reserved >= count)
       {
             memXor2(buf, st->block + 16 - st->reserved, count);
             st->reserved -= count;
             return;
       }
       memXor2(buf, st->block + 16 - st->reserved, st->reserved);
       count -= st->reserved;
       buf = (octet*)buf + st->reserved;
       st->reserved = 0;
 }
 // цикл по полным блокам
 while (count >= 16)
 {
       beltBlockIncU32(st->ctr);
       beltBlockCopy(st->block, st->ctr);
       beltBlockEncr((u32*)st->block, st->key);
       beltBlockXor2(buf, st->block);
       buf = (octet*)buf + 16;
       count -= 16;
 }
 // неполный блок?
 if (count)
 {
       beltBlockIncU32(st->ctr);
       beltBlockCopy(st->block, st->ctr);
       beltBlockEncr((u32*)st->block, st->key);
       memXor2(buf, st->block, count);
       st->reserved = 16 - count;
 }
}

//
//file dip_proj.ino
//

#include <SPI.h>
#include <Ethernet.h>
#include <EEPROM.h>
#include <SoftwareSerial.h>
#include <WiFiEsp.h>

#include <belt.h>


bool SPI_is_receiving = false; //индикация получения данных по SPI

#define KEY_LEN 16 //длина ключа
const octet iv[16] = { 'z', 'j', 'l', 'b', 'y', ':', 'b', 'd', '0', 'q',
'f', 'l', 'h', 'e', 'u', '7' }; //синхропосылка
```

```
    char outData[256] = {0}; // буфер для исходящих данных, нуждающиеся в
шифровании

    belt_ctr_st stateEncr = {0}; // состояние шифратора
    belt_ctr_st stateDecr = {0}; // состояние дешифратора

    // структура конфигурации модуля
    typedef struct _CONFIG {
        octet key[KEY_LEN]; // ключ шифрования
        char ssid[32]; // WiFi SSID
        char pass[32]; // пароль
        char server[64]; // адрес сервера для передачи данных
    } CONFIG;

    unsigned short out_counter = 0;
    unsigned short in_counter = 0;
    // Уникальный идентификатор клиента
    unsigned short cid = 0;

    /*
    Описание протокола обмена данными с сервером.
    Клиент обменивается данными с сервером при помощи пакетов, имеющих
следующую структуру:
    +--------------+-------------+--------------+------------------+------+
    | cid: 2 байта | cmd: 1 байт | len: 1 байта | counter: 2 байта | data |
    +--------------+-------------+--------------+------------------+------+

    cid - уникальный идентификатор клиента, назначается сервером при первом
подключении,
        если клиент обратился к нему с cid = 0 и cmd = CMD_HELLO
        Идентификатор клиента необходим для того, чтобы сервер мог сохранять
состояние шифраторов
        синхронизированное с клиентом.
    cmd - идетификатор команды:
        CMD_HELLO - отправляется клиентом при первом подключении для
получения cid
        CMD_SETCID - отправляется сервером для назначения клиенту cid
        CMD_DATA - пакет содержит полезные данные
    len - длина полезных данных
    counter - счетчик переданных полезных данных, необходим для синхронизации
шифраторов
    data - полезные данные (поле может быть пустым)
    */

    #define CMD_HELLO 0
    #define CMD_SETCID 1
    #define CMD_DATA 2
    typedef struct _DATA_PACKET {
        unsigned short cid; // уникальный идентификатор клиента
        octet cmd; // идентификатора команды
        octet len; // длина данных в пакете
        unsigned short counter; // счетчик количетсва переданных данных
    } DATA_PACKET;
```

```
    CONFIG Config = {0};
    int status = WL_IDLE_STATUS;     // the Wifi radio's status

    // программный UART для связи с WiFi модулем
    SoftwareSerial WifiSerial(6, 7); // RX, TX

    // Инициализация объекта клиента Ethernet
    WiFiEspClient client;

    unsigned long lastConnectionTime = 0;                 // время последнего
подключения к серверу, мс
    const unsigned long postingInterval = 10000L; // задержка между
подключениями к серверу, мс


    #define PIN_PD4 4
    #define PIN_LED_WIFI 3
    #define PIN_LED_COMM 5

    // вспомогательная функция для чтения строки из Serial
    void UtilSerialReadLine(char * out, int max_len)
    {
        for(i = 0; i < max_len; i++)
        {
            while(Serial.available() == 0){} // ожидание начала передачи
данных
            c = Serial.read(); // считывание строки посимвольно
            if (c == 10 || c == 13) // перевод на новую строку
                break; // выход
            out[i] = c;
        }
    }

    void setup() {
        int i; // счетчик
        char c, * ptr; // вспомогательные переменные

        pinMode(PIN_PD4, INPUT); // Кнопка PD4
        pinMode(PIN_LED_WIFI, OUTPUT); // Светодиод индикации подключения по
WiFi
        pinMode(PIN_LED_COMM, OUTPUT); // Светодиод индикации коммуникации
        digitalWrite(PIN_LED_WIFI, LOW);
        digitalWrite(PIN_LED_COMM, LOW);

        // инициализация SPI:
        SPI.begin();
        SPI.setBitOrder(MSBFIRST); // MSBFIRST – приоритет старшего бита,
LSBFIRST – приоритет младшего бита
        SPI.setClockDivider(SPI_CLOCK_DIV4); //установка делителя частоты SPI
(16Mhz/4 = 4Mhz)
        digitalWrite(SS, HIGH);

        // инициализация UART:
        Serial.begin(9600); // инициализация UART для настройки и отладки
        WifiSerial.begin(9600); // инициализация UART для связи с WiFi
```

```
        buttonState = digitalRead(4); // чтения состояния кнопки

        // Проверка, нажата ли кнопка PD4 сразу после включения питания
        if (buttonState == 0){ // Если нажата, необходимо ввести конфигурацию
(Wifi, ключ шифрования, адрес сервера)
            for(i = 0; i < sizeof(CONFIG); i++){
                EEPROM.write(i, 0); // Обнуляем EEPROM
            }

            Serial.print("Input key (16 bytes): ");
            while(Serial.available() <= 0){ ;} // Ожидание ввода
            delay(1000); // Задержка для передачи в буфер

            for(i = 0; i < KEY_LEN; i++) // считывание ключа с
последовательного порта при запуске
            {
                while(Serial.available() == 0){} // ожидание начала передачи
данных
                Config.key[i] = Serial.read(); // считывание ключа шифрования
посимвольно
            }

            Serial.println();
            Serial.print("Input SSID (32 bytes max, end with ENTER): ");
            UtilSerialReadLine(Config.ssid, 32); // ввод SSID

            Serial.println();
            Serial.print("Input password (32 bytes max, end with ENTER): ");
            UtilSerialReadLine(Config.pass, 32); // ввод пароля WiFi

            Serial.println();
            Serial.print("Input server address (64 bytes max, end with
ENTER): ");
            UtilSerialReadLine(Config.server, 64); // ввод пароля WiFi

            ptr = &Config;
            for(i = 0; i < sizeof(CONFIG); i++)
                EEPROM.write(i, ptr[i]); // Запись конфигурации в EEPROM

            Serial.println();
        }
        else {
            ptr = &Config;
            for(i = 0; i < sizeof(CONFIG); i++)
                ptr[i] = EEPROM.read(i); // чтение конфигурации из EEPROM
        }

        // инициализация WiFi модуля
        WiFi.init(&WifiSerial);

        // проверка наличия WiFI
        if (WiFi.status() == WL_NO_SHIELD) {
            Serial.println("WiFi shield not present");
```

```
            while (true); // без wifi невозможно продолжить работу, поэтому
остаемся в вечном цикле
        }

        // попытка подключения к WiFi сети
        while ( status != WL_CONNECTED) {
            Serial.print("Attempting to connect to WPA SSID: ");
            Serial.println(Config.ssid);
            // Подключение к сети WPA/WPA2
            status = WiFi.begin(Config.ssid, Config.pass);
        }

        // WiFi подключен
        Serial.println("You're connected to the network");
        digitalWrite(PIN_LED_WIFI, HIGH);

        // инициализация состояния шифратора
        beltCTRStart(&stateEncr, Config.key, KEY_LEN, iv);

        // инициализация состояния дешифратора
        beltCTRStart(&stateDecr, Config.key, KEY_LEN, iv);

        Serial.print("Server name: ");
        Serial.println(Config.server);
    }

    bool wait_data(unsigned int timeout) {
        unsigned int i;
        while(client.connected() && !client.available() && i < timeout) {
            delay(1); // 1ms
            i++;
        }
        return client.connected() && client.available();
    }

    size_t SPI_counter; // счетчик для SPI буфера
    bool SPI_available = false; // служит для индикации наличия полезныех
данных принимаемых по SPI

    bool SPI_Read(char input) //обработка приёма по SPI
    {
        if(!SPI_is_receiving)    //если    передача    полезных    данных    не
осуществляется
        {
            if(input == 0xFF) // незначащий байт заполнения
            {
                SPI_available = false;
                return;
            }

            if(input  ==  1)  //сигнализация,  что  далее  будет  осуществляться
передача полезных данных
            {
                SPI_is_receiving = true;
                SPI_available = true;
```

```
                return;
            }
        }

        SPI_is_receiving = false;
        if(SPI_counter == 256)
            SPI_counter = 0;
        outData[SPI_counter] = input;
        SPI_counter ++;

        return SPI_available;
    }

    void SPI_Transfer(octet data) {
        SPI_Read(SPI.transfer(1)); // сигнализация того, что следующий байт
содержит полезные данные
        SPI_Read(SPI.transfer(outData));
    }

    void loop() {
        DATA_PACKET pkt = {0};
        octet * ptr = &pkt;
        int i;
        octet c;

        // если cid не установлен, то нужно отправить первый запрос на сервер
        if(cid == 0) {
            pkt.cmd = CMD_HELLO;
            pkt.len = 0;
            pkt.cid = 0;
            pkt.counter = 0;
            SendData(&pkt, outData);
        }

        // Получить данные от сервера, если соедининение установлено и есть
данные
        if(wait_data(1000)) {
            // получение заголовка
            for(i = 0; i < sizeof(DATA_PACKET); i++) {
                ptr[i] = client.read();
            }

            if(pkt.cmd == CMD_SETCID) {
                // данные в пакете содержат новую синхропосылку
                if(pkt.len == 16)
                    for(i = 0; i < 16; i++)
                        iv[i] = client.read();
                // переинициализация шифратора
                beltCTRStart(&stateEncr, Config.key, KEY_LEN, iv);
                beltCTRStart(&stateDecr, Config.key, KEY_LEN, iv);
                out_counter = 0;
                in_counter = 0;
                cid = pkt.cid;
            }
```

```
            //  если  произошла  рассинхронизация,  необходимо  разорвать
соединение
            // и заново проинициализировать шифратор
            if(in_counter + pkt.len != pkt.counter) {
                cid = 0;
                client.stop();
                return;
            }

            // получение данных
            for(i = 0; i < pkt.len; i++) {
                c = (octet)client.read();
                beltCTRStepE(&c, 1, &stateDecr); //расшифровка байта данных
                in_counter++;

                SPI_Transfer(c);
            }
        }

        // получить данные от устройства по SPI, зашифровать и отправить на
сервер
        if(SPI_Read(0xFF) || SPI_counter > 0)
        {
            // читаем, пока есть данные или буфер не заполнится
            while(SPI_Read(0xFF) && SPI_counter < 256);

            // шифрование данных, полученных по SPI и отправка на сервер
            beltCTRStepE(outData, SPI_counter, &stateEncr);
            out_counter += SPI_counter;

            // формирование заголовка
            pkt.cmd = CMD_DATA;
            pkt.cid = cid;
            pkt.len = SPI_counter;
            pkt.counter = out_counter;
            SPI_counter = 0;

            // отправка данных
            SendData(&pkt, outData);
            return;
        }

        // через кадые 10 секунд опрашиваем сервер даже если нам нечего ему
отправить
        if (millis() - lastConnectionTime > postingInterval) {
            pkt.cmd = CMD_DATA;
            pkt.cid = cid;
            pkt.len = 0;
            pkt.counter = out_counter;
            SendData(&pkt, &outData);
        }
    }

    // отправка данных на сервер
    void SendData(DATA_PACKET * pkt, octet * data)
```

```
{
    size_t i;
    Serial.println();

    // закрываем прошлые соединения перед отправкой нового запроса,
    // это освободит сокет на WiFi модуле
    client.stop();

    // пытаемся подключиться к серверу на 8888 порт
    if (client.connect(server, 8888)) {
        Serial.println("Connecting...");
        digitalWrite(PIN_LED_COMM, HIGH);

        // отправка заголовка и данных
        client.write(pkt, sizeof(DATA_PACKET));
        if(pkt->len > 0)
            client.write(data, pkt->len);

        // note the time that the connection was made
        lastConnectionTime = millis();

        // отладочный вывод заголовка и данных
        Serial.print(postRequest);
        Serial.write(data, length);
    }
    else {
        // подключение не удалось
        Serial.println("Connection failed");
    }
}
```