ONLINE CARD PROTOCOL

Сеансовый уровень

Оглавление

1. Общие сведения 3

2. Управляющие символы протокола 3

3. Состояния логического соединения 3

3.1. Установление соединения 3
2.1. Обмен данными 6
2.2. Удержание соединения 9
2.3. Завершение соединения 10

Приложение 1 11

Реализация вычисления CRC-8 11
Реализация вычисления CRC-16 11
Реализация вычисления CRC-32 12
Реализация вычисления CRC-64 12

1. ОБЩИЕ СВЕДЕНИЯ

Протокол ОСР является байт-ориентированным протоколом сеансового уровня, разработанным в соответствии с рекомендациями RFC1547, RFC1549. Данный протокол соответствует пятому уровню в модели OSI и обеспечивает управление логическим соединением с гарантированной передачей данных. В протокол встроен контроль ошибок передачи и возможности шифрования трафика.

2. УПРАВЛЯЮЩИЕ СИМВОЛЫ ПРОТОКОЛА

Пакетная передача данных, которая лежит в основе протокола ОСР, осуществляется с использованием служебных символов, перечисленных в табл. «Служебные символы протокола ОСР».

Таблица Служебные символы протокола ОСР

Название	HEX-код	Назначение
ENQ	0x05	Запрос на установление логического соединения
ACK	0x06	Символ, подтверждающий успешное получение: • запроса на соединение • пакета данных
EOT	0x04	Символ закрытия логического соединения
STX	0x02	Указывает на начало пакета данных
ETX	0x03	Указывает на конец пакета данных
BEL	0x07	Символ, использующийся для удержания соединения
NAK	0x15	Символ уведомляет передающую сторону о том, что принимаемая сторона получила искаженные данные и ожидает повторной передачи данных.
TMC	0x88	Символ уведомляет клиента о том, что сервер перегружен, и нужно попытаться соединиться с другим сервером. Высылается только в ответ на ENQ.

3. СОСТОЯНИЯ ЛОГИЧЕСКОГО СОЕДИНЕНИЯ

Логическое соединение может находиться в четырёх основных состояниях:

- Установление соединения
- Обмен данными
- Удержание соединения
- Завершение соединения

3.1. Установление соединения

Для организации процесса приёма/передачи данных по протоколу ОСР, обоим участникам необходимо установить логическое соединение. Процесс установки соединения зависит от того, по каким принципам будет происходить обмен данными и будет ли зашифрован канал связи. Инициатор соединения (Клиент) последовательно отправляет приёмнику (Серверу) три байта:

1. байт со служебным символом ENQ;
2. байт с версией протокола (далее ProtocolVersion или PV). Протокол определяет алгоритм и параметры шифрования сессионной информации, размер сессионного ключа;
3. байт с версией набора ключей шифрования (далее CipherVersion или CV).

Сервер, после получения запроса на соединение, отправляет в ответ следующую информацию:

1. байт со служебным символом ACK;
2. служебную информацию, зависящую от PV и CV (далее ConnectionInfo или CI). CI может не отправляться, если протокол не использует шифрование данных.

В версии протокола с шифрованием используются три ключа:

1. транспортный ключ (TK);

2. ключ клиента (CK);
3. Сессионный ключ для шифрования данных внутри сессии (SK)

Размер ConnectionInfo и параметры используемых криптоалгоритмов приведены в таблице:

PV	CV	Размер CI, байт	Шифрование поля Data	Шифрование CI	Размер SK, бит	Размер TK, бит	Размер CK, бит
0	0	0	-	-		-	-
1	0..FF	8	DES ECB	3DES ECB	64	192	192
2	0..FF	16	3DES ECB	3DES ECB	128	192	192
3	0..FF	16	AES CBC	AES CBC	128	256	256
4	0..FF	32	AES CBC	AES CBC	256	256	256

Все алгоритмы шифрования из таблицы не используют вектор инициализации (он нулевой) и не используют выравнивание данных, т.к. оно предусмотрено текущим протоколом.

У каждой версии протокола наборы ключей CV=0 считаются стандартными: для них имеются предустановленные и определённые ключи шифрования.

Формирование ConnectionInfo и генерация сессионного ключа

Пусть N: это размер ConnectionInfo в байтах, тогда алгоритм генерации сессионного ключа выглядит следующим образом:

1. Сервер должен выполнить следующие действия:
   - проверить PV и CV, предложенные клиентом, загрузить требуемые ключи и инициализировать криптоалгоритмы;
   - сгенерировать N-1 случайных байт RND[1]…RND[N-1]. Желательно если будет генерироваться «сильная» выборка, с точки зрения криптостойкости;
   - вычислить CRC8 (см. приложение 1) для массива RND[1]…RND[N-1];
   - получить 16 байт незашифрованного CI добавив CRC8 к массиву следующим образом:

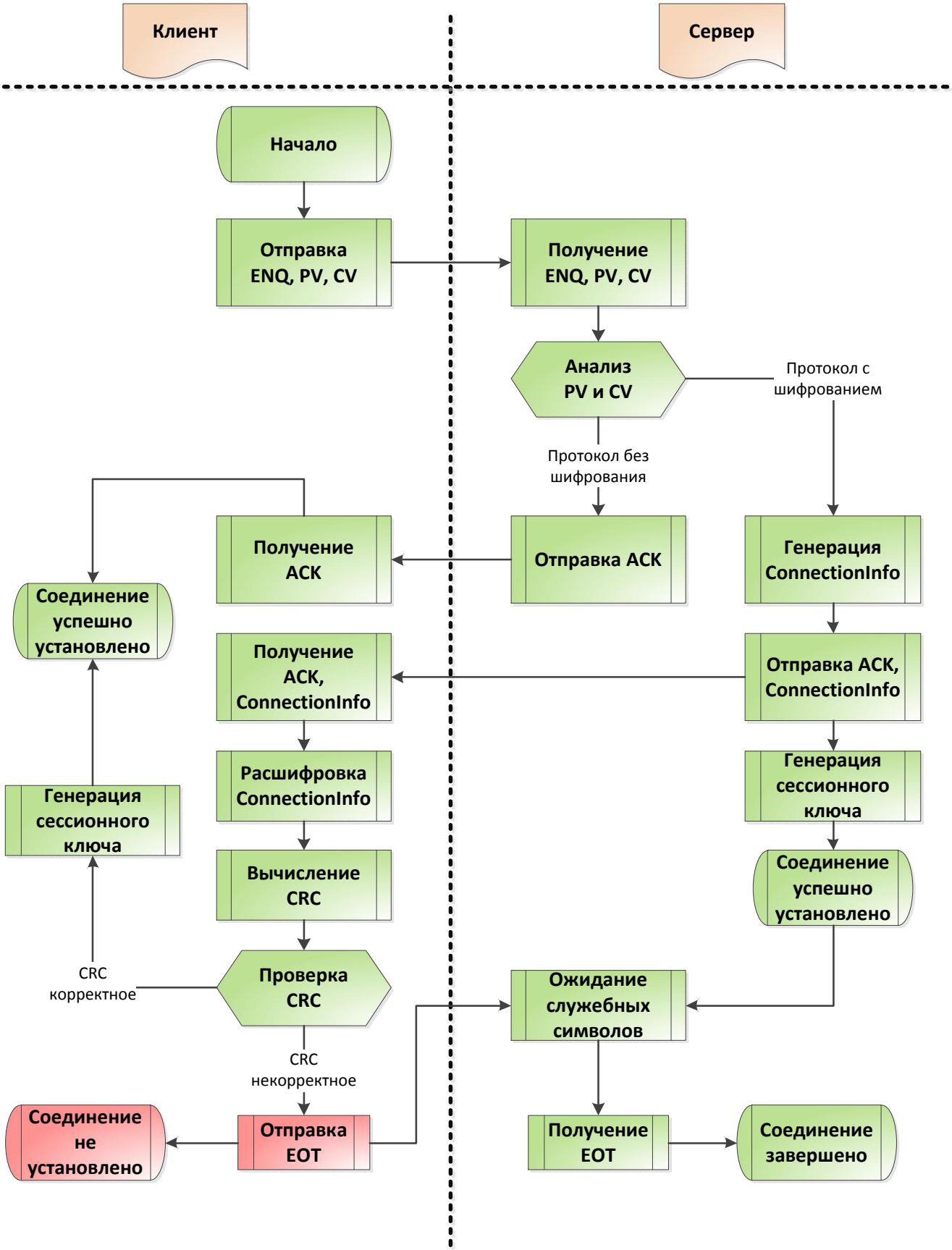
RND[1]	RND[2]	…	RND[N-1]	CRC8

   - зашифровать N байт CI по алгоритму шифрования ConnectionInfo транспортным ключом;
   - отправить клиенту ответ на соединение:

ACK	N байт ConnectionInfo

2. Клиент, после получения ответа от сервера, должен выполнить следующие действия:
   - проверить ACK;
   - расшифровать транспортным ключом полученные данные;
   - вычислить CRC8 для RND[1]…RND[N-1];
   - сравнить вычисленное CRC8 и полученным. В случае несовпадения, отправить серверу символ EOT для завершения соединения;
   - получить сессионный ключ, зашифровав N байт «RND1…RND[N-1], CRC8» ключом клиента по алгоритму шифрования ConnectionInfo. Сервер получает ключ аналогичным образом;
   - запомнить полученный сессионный ключ в энергозависимой защищённой памяти.

Схема процесса установления соединения:


Клиент
Сервер
Начало
Отправка ENQ, PV, CV
Получение ENQ, PV, CV
Анализ PV и CV
Протокол с шифрованием
Протокол без шифрования
Получение ACK
Отправка ACK
Генерация ConnectionInfo
Соединение успешно установлено
Получение ACK, ConnectionInfo
Отправка ACK, ConnectionInfo
Расшифровка ConnectionInfo
Генерация сессионного ключа
Генерация сессионного ключа
Вычисление CRC
Соединение успешно установлено
CRC корректное
Проверка CRC
Ожидание служебных символов
CRC некорректное
Соединение не установлено
Отправка EOT
Получение EOT
Соединение завершено

3.2. Обмен данными

Обмен данными по протоколу OCP осуществляется пакетами, состоящими из следующих полей:

STX	Data Size	Data	ETX	CRC

Где:

1. Служебный символ **STX**: указатель на начало пакета
2. **Data Size**: поле хранит невыровненную длину передаваемых данных (размер поля Data). Поле переменной длины (1, 2, 3 или 5 байт). Формат поля, методы его построения и чтения описаны в документе «OCP 2. Представление данных TLV» раздел «Описание поля Length».
3. **Data**: Передаваемые данные: байтовый массив с данными, размер которых указан в поле Data Size. В случае незашифрованного трафика Data Size и размер поля Data совпадают. В случае зашифрованного трафика, Data Size указывает реальный размер данных, но в коммуникационный канал должен быть записан блок данных, выровненный в соответствии с используемым алгоритмом шифрования. Обозначим размер данного блока как Aligned Data Size (ADS). Данное число всегда ≥ Data Size. Вычисляется оно следующим образом:

PV	CV	Align	Вычисление ADS	Пример
0	0	-	ADS = Data Size	
1	любой	8 байт	M = Data Size mod Align. Если M ≠ 0, то ADS = Data Size + Align – M, Если M = 0, то ADS = Data Size.	Для Data Size = 11: ADS = 11+8-(11 mod 8) = 16. Для Data Size = 8: ADS = 8.
2				
3	любой	16 байт		Для Data Size = 28: ADS = 28+16-(28 mod 16) = 32. Для Data Size = 16: ADS = 16.
4				

4. Служебный символ **ETX**: указатель на завершение блока данных (разделитель блока данных и CRC)
5. **CRC**: контрольная сумма: поле переменной длины, в котором находится беззнаковое целое число, записанное в обратной последовательности байт (старший–младший). Контрольные суммы вычисляются по алгоритмам, приведённых в приложении №1 для набора данных, состоящих из полей: **CRCBuffer** = [**Data Size, Data, ETX**], которые выделены на схеме.
   Размер поля CRC и алгоритм нахождения контрольной суммы зависят от размера CRCBuffer:

Размер CRCBuffer, байт	Размер поля CRC, байт	Алгоритм вычисления CRC
< 64 (< 40h)	1	CRC8
64..16384 (40h..4000h)	2	CRC16
16385..536870912 (4001h..20000000h)	4	CRC32
> 536870912 (> 20000000h)	8	CRC64

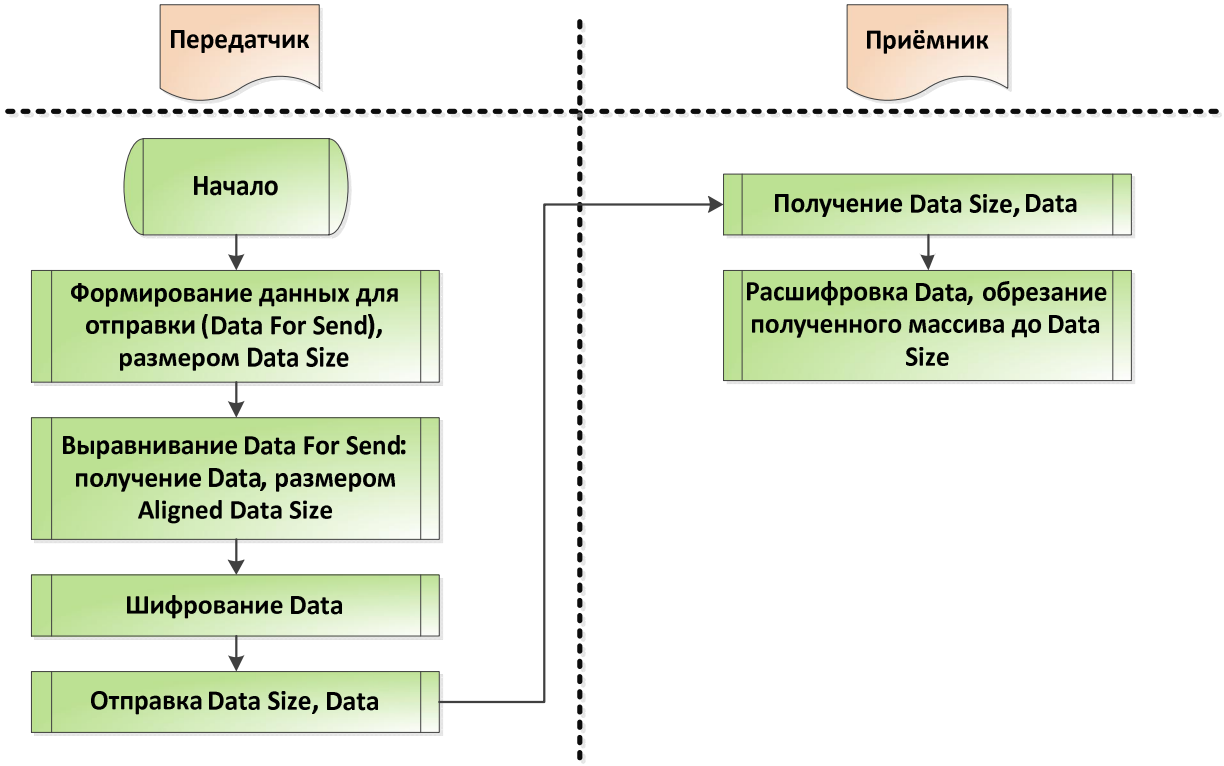
При использовании шифрования, данные перед отправкой должны быть выровнены. При дополнении исходных данных выравнивающими, рекомендуется генерировать псевдослучайные байты. Например, для Data Size = 11 (Aligned Data Size = 16):

Исходные данные											Дополнение к данным				
D1	D2	D3	D4	D5	D6	D7	D8	D9	D10	D11	RND1	RND2	RND3	RND4	RND5

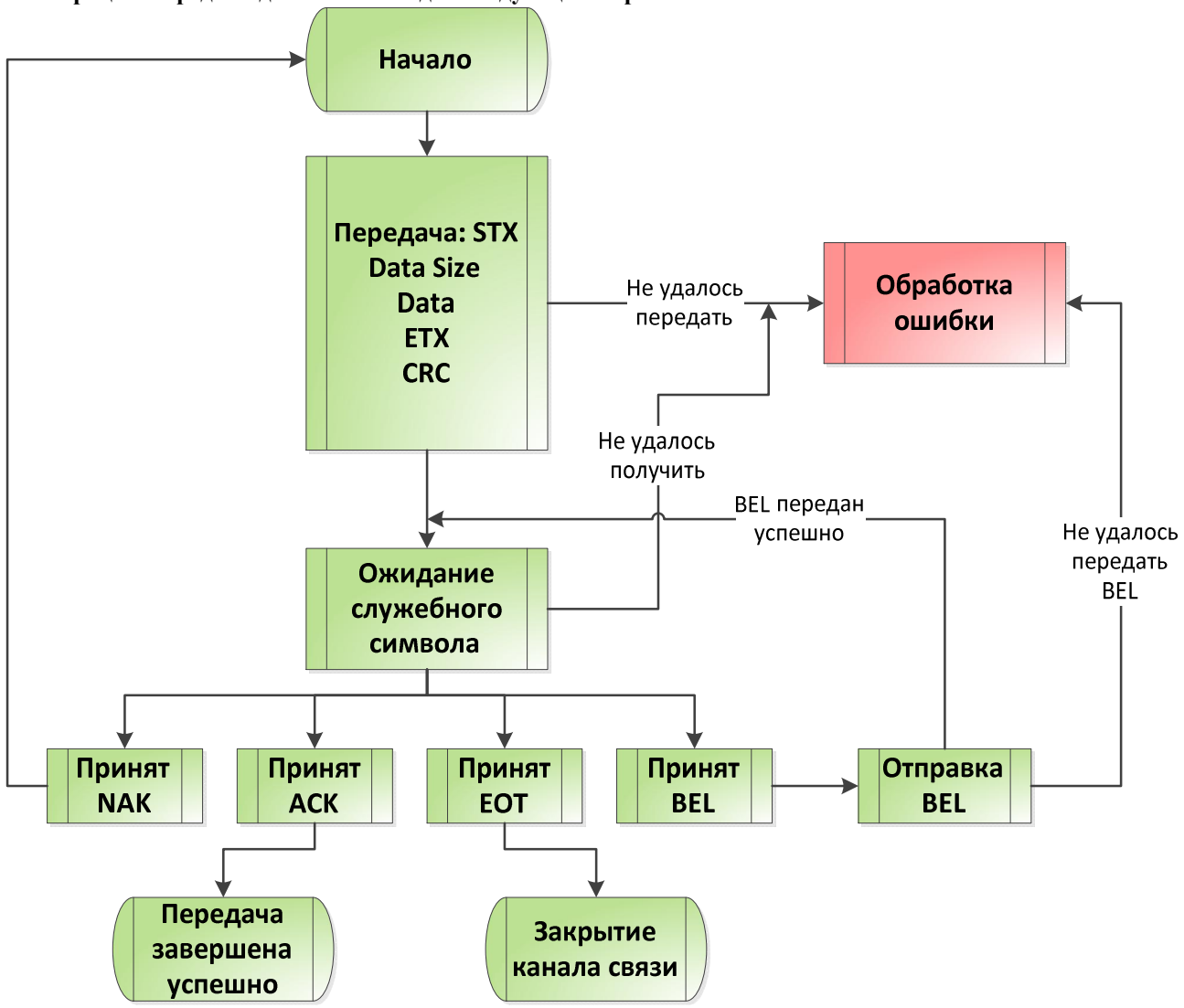
После выравнивания, поле Data шифруется выбранным алгоритмом шифрования с использованием сессионного ключа.

Преобразование данных после получения выполняется в обратном порядке: Data расшифровывается, после чего обрезается до размера Data Size.

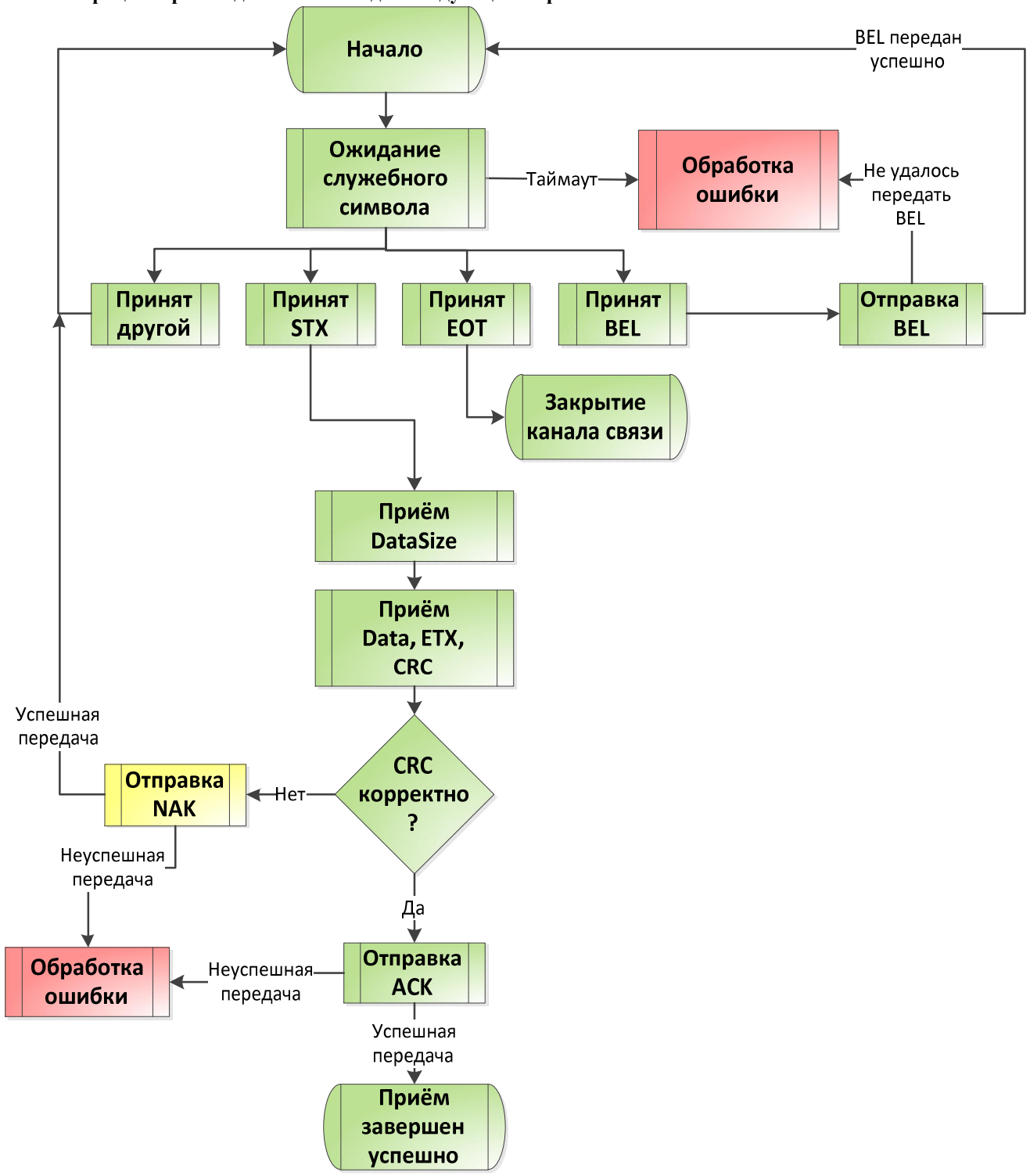
Процесс преобразования данных и обмена показан на схеме:


Передатчик
Приёмник
Начало
Формирование данных для отправки (Data For Send), размером Data Size
Выравнивание Data For Send: получение Data, размером Aligned Data Size
Шифрование Data
Отправка Data Size, Data
Получение Data Size, Data
Расшифровка Data, обрезание полученного массива до Data Size

Процесс передачи данных выглядит следующим образом:



Процесс приёма данных выглядит следующим образом:



3.3. Удержание соединения

После соединения участников обмена по протоколу ОСР возможны случаи, когда один из участников не отправляет долгое время пакеты другому (например, во время длительных операций). Для того, чтобы избежать таймаута при приёме (и последующего разрыва соединения) участники должны уведомлять друг друга о длительных операциях и выполнять процедуру удержания соединения. Для этого один из участников отправляет другому служебный символ BEL. Другой участник обмена, при получении данного символа должен также ответить символом BEL и обнулить таймер таймаута.

Процесс удержания соединения показан на схемах приёма и передачи данных.
Символы BEL не должны отправляться в середине пакета с данными.

3.4. Завершение соединения

После завершения обмена данными, одна из сторон уведомляет другую о завершении логического соединения, отправляя в коммуникационный канал символ EOT. При получении данного символа другой участник должен прекратить операции ожидания или передачи данных и закрыть коммуникационный канал.

4. ПРИЛОЖЕНИЕ 1.

В приложении приведены реализации алгоритмов вычисления контрольных сумм для компилятора GNU C.

Реализация вычисления CRC-8

```
static const unsigned char Crc8Table[256] = {
    0x00, 0x31, 0x62, 0x53, 0xC4, 0xF5, 0xA6, 0x97, 0xB9, 0x88, 0xDB, 0xEA, 0x7D, 0x4C, 0x1F, 0x2E,
    0x43, 0x72, 0x21, 0x10, 0x87, 0xB6, 0xE5, 0xD4, 0xFA, 0xCB, 0x98, 0xA9, 0x3E, 0x0F, 0x5C, 0x6D,
    0x86, 0xB7, 0xE4, 0xD5, 0x42, 0x73, 0x20, 0x11, 0x3F, 0x0E, 0x5D, 0x6C, 0xFB, 0xCA, 0x99, 0xA8,
    0xC5, 0xF4, 0xA7, 0x96, 0x01, 0x30, 0x63, 0x52, 0x7C, 0x4D, 0x1E, 0x2F, 0xB8, 0x89, 0xDA, 0xEB,
    0x3D, 0x0C, 0x5F, 0x6E, 0xF9, 0xC8, 0x9B, 0xAA, 0x84, 0xB5, 0xE6, 0xD7, 0x40, 0x71, 0x22, 0x13,
    0x7E, 0x4F, 0x1C, 0x2D, 0xBA, 0x8B, 0xD8, 0xE9, 0xC7, 0xF6, 0xA5, 0x94, 0x03, 0x32, 0x61, 0x50,
    0xBB, 0x8A, 0xD9, 0xE8, 0x7F, 0x4E, 0x1D, 0x2C, 0x02, 0x33, 0x60, 0x51, 0xC6, 0xF7, 0xA4, 0x95,
    0xF8, 0xC9, 0x9A, 0xAB, 0x3C, 0x0D, 0x5E, 0x6F, 0x41, 0x70, 0x23, 0x12, 0x85, 0xB4, 0xE7, 0xD6,
    0x7A, 0x4B, 0x18, 0x29, 0xBE, 0x8F, 0xDC, 0xED, 0xC3, 0xF2, 0xA1, 0x90, 0x07, 0x36, 0x65, 0x54,
    0x39, 0x08, 0x5B, 0x6A, 0xFD, 0xCC, 0x9F, 0xAE, 0x80, 0xB1, 0xE2, 0xD3, 0x44, 0x75, 0x26, 0x17,
    0xFC, 0xCD, 0x9E, 0xAF, 0x38, 0x09, 0x5A, 0x6B, 0x45, 0x74, 0x27, 0x16, 0x81, 0xB0, 0xE3, 0xD2,
    0xBF, 0x8E, 0xDD, 0xEC, 0x7B, 0x4A, 0x19, 0x28, 0x06, 0x37, 0x64, 0x55, 0xC2, 0xF3, 0xA0, 0x91,
    0x47, 0x76, 0x25, 0x14, 0x83, 0xB2, 0xE1, 0xD0, 0xFE, 0xCF, 0x9C, 0xAD, 0x3A, 0x0B, 0x58, 0x69,
    0x04, 0x35, 0x66, 0x57, 0xC0, 0xF1, 0xA2, 0x93, 0xBD, 0x8C, 0xDF, 0xEE, 0x79, 0x48, 0x1B, 0x2A,
    0xC1, 0xF0, 0xA3, 0x92, 0x05, 0x34, 0x67, 0x56, 0x78, 0x49, 0x1A, 0x2B, 0xBC, 0x8D, 0xDE, 0xEF,
    0x82, 0xB3, 0xE0, 0xD1, 0x46, 0x77, 0x24, 0x15, 0x3B, 0x0A, 0x59, 0x68, 0xFF, 0xCE, 0x9D, 0xAC
};

unsigned char Crc8(unsigned char *pcBlock, unsigned char len)
{
    unsigned char crc = 0xFF;
    while (len--)
        crc = Crc8Table[crc ^ *pcBlock++];
    return crc;
}
```

Реализация вычисления CRC-16

```
const unsigned short Crc16Table[256] = {
        0x0000, 0x1021, 0x2042, 0x3063, 0x4084, 0x50A5, 0x60C6, 0x70E7,
        0x8108, 0x9129, 0xA14A, 0xB16B, 0xC18C, 0xD1AD, 0xE1CE, 0xF1EF,
        0x1231, 0x0210, 0x3273, 0x2252, 0x52B5, 0x4294, 0x72F7, 0x62D6,
        0x9339, 0x8318, 0xB37B, 0xA35A, 0xD3BD, 0xC39C, 0xF3FF, 0xE3DE,
        0x2462, 0x3443, 0x0420, 0x1401, 0x64E6, 0x74C7, 0x44A4, 0x5485,
        0xA56A, 0xB54B, 0x8528, 0x9509, 0xE5EE, 0xF5CF, 0xC5AC, 0xD58D,
        0x3653, 0x2672, 0x1611, 0x0630, 0x76D7, 0x66F6, 0x5695, 0x46B4,
        0xB75B, 0xA77A, 0x9719, 0x8738, 0xF7DF, 0xE7FE, 0xD79D, 0xC7BC,
        0x48C4, 0x58E5, 0x6886, 0x78A7, 0x0840, 0x1861, 0x2802, 0x3823,
        0xC9CC, 0xD9ED, 0xE98E, 0xF9AF, 0x8948, 0x9969, 0xA90A, 0xB92B,
        0x5AF5, 0x4AD4, 0x7AB7, 0x6A96, 0x1A71, 0x0A50, 0x3A33, 0x2A12,
        0xDBFD, 0xCBDC, 0xFBBF, 0xEB9E, 0x9B79, 0x8B58, 0xBB3B, 0xAB1A,
        0x6CA6, 0x7C87, 0x4CE4, 0x5CC5, 0x2C22, 0x3C03, 0x0C60, 0x1C41,
        0xEDAE, 0xFD8F, 0xCDEC, 0xDDCD, 0xAD2A, 0xBD0B, 0x8D68, 0x9D49,
        0x7E97, 0x6EB6, 0x5ED5, 0x4EF4, 0x3E13, 0x2E32, 0x1E51, 0x0E70,
        0xFF9F, 0xEFBE, 0xDFDD, 0xCFFC, 0xBF1B, 0xAF3A, 0x9F59, 0x8F78,
        0x9188, 0x81A9, 0xB1CA, 0xA1EB, 0xD10C, 0xC12D, 0xF14E, 0xE16F,
        0x1080, 0x00A1, 0x30C2, 0x20E3, 0x5004, 0x4025, 0x7046, 0x6067,
        0x83B9, 0x9398, 0xA3FB, 0xB3DA, 0xC33D, 0xD31C, 0xE37F, 0xF35E,
        0x02B1, 0x1290, 0x22F3, 0x32D2, 0x4235, 0x5214, 0x6277, 0x7256,
        0xB5EA, 0xA5CB, 0x95A8, 0x8589, 0xF56E, 0xE54F, 0xD52C, 0xC50D,
        0x34E2, 0x24C3, 0x14A0, 0x0481, 0x7466, 0x6447, 0x5424, 0x4405,
        0xA7DB, 0xB7FA, 0x8799, 0x97B8, 0xE75F, 0xF77E, 0xC71D, 0xD73C,
        0x26D3, 0x36F2, 0x0691, 0x16B0, 0x6657, 0x7676, 0x4615, 0x5634,
        0xD94C, 0xC96D, 0xF90E, 0xE92F, 0x99C8, 0x89E9, 0xB98A, 0xA9AB,
        0x5844, 0x4865, 0x7806, 0x6827, 0x18C0, 0x08E1, 0x3882, 0x28A3,
        0xCB7D, 0xDB5C, 0xEB3F, 0xFB1E, 0x8BF9, 0x9BD8, 0xABBB, 0xBB9A,
        0x4A75, 0x5A54, 0x6A37, 0x7A16, 0x0AF1, 0x1AD0, 0x2AB3, 0x3A92,
        0xFD2E, 0xED0F, 0xDD6C, 0xCD4D, 0xBDAA, 0xAD8B, 0x9DE8, 0x8DC9,
        0x7C26, 0x6C07, 0x5C64, 0x4C45, 0x3CA2, 0x2C83, 0x1CE0, 0x0CC1,
        0xEF1F, 0xFF3E, 0xCF5D, 0xDF7C, 0xAF9B, 0xBFBA, 0x8FD9, 0x9FF8,
        0x6E17, 0x7E36, 0x4E55, 0x5E74, 0x2E93, 0x3EB2, 0x0ED1, 0x1EF0
```

```
};

unsigned short Crc16(unsigned char *pcBlock, unsigned int len)
{
        unsigned short crc = 0xFFFF;
        while (len--)
                crc = (crc << 8) ^ Crc16Table[(crc >> 8) ^ *pcBlock++];
        return crc;
}
```

Реализация вычисления CRC-32

```
/*
  Name  : CRC-32
  Poly  : 0x04C11DB7    x^32 + x^26 + x^23 + x^22 + x^16 + x^12 + x^11
                       + x^10 + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1
  Init  : 0xFFFFFFFF
  Revert: true
  XorOut: 0xFFFFFFFF
  Check : 0xCBF43926 ("123456789")
  MaxLen: 268 435 455 байт (2 147 483 647 бит) - обнаружение
   одинарных, двойных, пакетных и всех нечетных ошибок
*/
static const unsigned int Crc32Table[256] = {
0x00000000, 0x77073096, 0xEE0E612C, 0x990951BA, 0x076DC419, 0x706AF48F, 0xE963A535, 0x9E6495A3, 0x0EDB8832,
0x79DCB8A4, 0xE0D5E91E, 0x97D2D988, 0x09B64C2B, 0x7EB17CBD, 0xE7B82D07, 0x90BF1D91, 0x1DB71064, 0x6AB020F2,
0xF3B97148, 0x84BE41DE, 0x1ADAD47D, 0x6DDDE4EB, 0xF4D4B551, 0x83D385C7, 0x136C9856, 0x646BA8C0, 0xFD62F97A,
0x8A65C9EC, 0x14015C4F, 0x63066CD9, 0xFA0F3D63, 0x8D080DF5, 0x3B6E20C8, 0x4C69105E, 0xD56041E4, 0xA2677172,
0x3C03E4D1, 0x4B04D447, 0xD20D85FD, 0xA50AB56B, 0x35B5A8FA, 0x42B2986C, 0xDBBBC9D6, 0xACBCF940, 0x32D86CE3,
0x45DF5C75, 0xDCD60DCF, 0xABD13D59, 0x26D930AC, 0x51DE003A, 0xC8D75180, 0xBFD06116, 0x21B4F4B5, 0x56B3C423,
0xCFBA9599, 0xB8BDA50F, 0x2802B89E, 0x5F058808, 0xC60CD9B2, 0xB10BE924, 0x2F6F7C87, 0x58684C11, 0xC1611DAB,
0xB6662D3D, 0x76DC4190, 0x01DB7106, 0x98D220BC, 0xEFD5102A, 0x71B18589, 0x06B6B51F, 0x9FBFE4A5, 0xE8B8D433,
0x7807C9A2, 0x0F00F934, 0x9609A88E, 0xE10E9818, 0x7F6A0DBB, 0x086D3D2D, 0x91646C97, 0xE6635C01, 0x6B6B51F4,
0x1C6C6162, 0x856530D8, 0xF262004E, 0x6C0695ED, 0x1B01A57B, 0x8208F4C1, 0xF50FC457, 0x65B0D9C6, 0x12B7E950,
0x8BBEB8EA, 0xFCB9887C, 0x62DD1DDF, 0x15DA2D49, 0x8CD37CF3, 0xFBD44C65, 0x4DB26158, 0x3AB551CE, 0xA3BC0074,
0xD4BB30E2, 0x4ADFA541, 0x3DD895D7, 0xA4D1C46D, 0xD3D6F4FB, 0x4369E96A, 0x346ED9FC, 0xAD678846, 0xDA60B8D0,
0x44042D73, 0x33031DE5, 0xAA0A4C5F, 0xDD0D7CC9, 0x5005713C, 0x270241AA, 0xBE0B1010, 0xC90C2086, 0x5768B525,
0x206F85B3, 0xB966D409, 0xCE61E49F, 0x5EDEF90E, 0x29D9C998, 0xB0D09822, 0xC7D7A8B4, 0x59B33D17, 0x2EB40D81,
0xB7BD5C3B, 0xC0BA6CAD, 0xEDB88320, 0x9ABFB3B6, 0x03B6E20C, 0x74B1D29A, 0xEAD54739, 0x9DD277AF, 0x04DB2615,
0x73DC1683, 0xE3630B12, 0x94643B84, 0x0D6D6A3E, 0x7A6A5AA8, 0xE40ECF0B, 0x9309FF9D, 0x0A00AE27, 0x7D079EB1,
0xF00F9344, 0x8708A3D2, 0x1E01F268, 0x6906C2FE, 0xF762575D, 0x806567CB, 0x196C3671, 0x6E6B06E7, 0xFED41B76,
0x89D32BE0, 0x10DA7A5A, 0x67DD4ACC, 0xF9B9DF6F, 0x8EBEEFF9, 0x17B7BE43, 0x60B08ED5, 0xD6D6A3E8, 0xA1D1937E,
0x38D8C2C4, 0x4FDFF252, 0xD1BB67F1, 0xA6BC5767, 0x3FB506DD, 0x48B2364B, 0xD80D2BDA, 0xAF0A1B4C, 0x36034AF6,
0x41047A60, 0xDF60EFC3, 0xA867DF55, 0x316E8EEF, 0x4669BE79, 0xCB61B38C, 0xBC66831A, 0x256FD2A0, 0x5268E236,
0xCC0C7795, 0xBB0B4703, 0x220216B9, 0x5505262F, 0xC5BA3BBE, 0xB2BD0B28, 0x2BB45A92, 0x5CB36A04, 0xC2D7FFA7,
0xB5D0CF31, 0x2CD99E8B, 0x5BDEAE1D, 0x9B64C2B0, 0xEC63F226, 0x756AA39C, 0x026D930A, 0x9C0906A9, 0xEB0E363F,
0x72076785, 0x05005713, 0x95BF4A82, 0xE2B87A14, 0x7BB12BAE, 0x0CB61B38, 0x92D28E9B, 0xE5D5BE0D, 0x7CDCEFB7,
0x0BDBDF21, 0x86D3D2D4, 0xF1D4E242, 0x68DDB3F8, 0x1FDA836E, 0x81BE16CD, 0xF6B9265B, 0x6FB077E1, 0x18B74777,
0x88085AE6, 0xFF0F6A70, 0x66063BCA, 0x11010B5C, 0x8F659EFF, 0xF862AE69, 0x616BFFD3, 0x166CCF45, 0xA00AE278,
0xD70DD2EE, 0x4E048354, 0x3903B3C2, 0xA7672661, 0xD06016F7, 0x4969474D, 0x3E6E77DB, 0xAED16A4A, 0xD9D65ADC,
0x40DF0B66, 0x37D83BF0, 0xA9BCAE53, 0xDEBB9EC5, 0x47B2CF7F, 0x30B5FFE9, 0xBDBDF21C, 0xCABAC28A, 0x53B39330,
0x24B4A3A6, 0xBAD03605, 0xCDD70693, 0x54DE5729, 0x23D967BF, 0xB3667A2E, 0xC4614AB8, 0x5D681B02, 0x2A6F2B94,
0xB40BBE37, 0xC30C8EA1, 0x5A05DF1B, 0x2D02EF8D
};

unsigned int Crc32(const unsigned char *pcBlock, unsigned int len)
{
    unsigned int crc = 0xFFFFFFFF;
    while (len--)
        crc = (crc >> 8) ^ Crc32Table[(crc ^ *pcBlock++) & 0xFF];
    return crc ^ 0xFFFFFFFF;
}
```

Реализация вычисления CRC-64

```
static uint64 Crc64Table[256] = {
0x0000000000000000ULL, 0x0809E8A2969451E9ULL, 0x1013D1452D28A3D2ULL, 0x181A39E7BBBCF23BULL,
0x2027A28A5A5147A4ULL, 0x282E4A28CCC5164DULL, 0x303473CF7779E476ULL, 0x383D9B6DE1EDB59FULL,
```

```
0x404F4514B4A28F48ULL, 0x4846ADB62236DEA1ULL, 0x505C9451998A2C9AULL, 0x58557CF30F1E7D73ULL,
0x6068E79EEEF3C8ECULL, 0x68610F3C78679905ULL, 0x707B36DBC3DB6B3EULL, 0x7872DE79554F3AD7ULL,
0x809E8A2969451E90ULL, 0x8897628BFFD14F79ULL, 0x908D5B6C446DBD42ULL, 0x9884B3CED2F9ECABULL,
0xA0B928A333145934ULL, 0xA8B0C001A58008DDULL, 0xB0AAF9E61E3CFAE6ULL, 0xB8A3114488A8AB0FULL,
0xC0D1CF3DDDE791D8ULL, 0xC8D8279F4B73C031ULL, 0xD0C21E78F0CF320AULL, 0xD8CBF6DA665B63E3ULL,
0xE0F66DB787B6D67CULL, 0xE8FF851511228795ULL, 0xF0E5BCF2AA9E75AEULL, 0xF8EC54503C0A2447ULL,
0x24B1909974C84E69ULL, 0x2CB8783BE25C1F80ULL, 0x34A241DC59E0EDBBULL, 0x3CABA97ECF74BC52ULL,
0x049632132E9909CDULL, 0x0C9FDAB1B80D5824ULL, 0x1485E35603B1AA1FULL, 0x1C8C0BF49525FBF6ULL,
0x64FED58DC06AC121ULL, 0x6CF73D2F56FE90C8ULL, 0x74ED04C8ED4262F3ULL, 0x7CE4EC6A7BD6331AULL,
0x44D977079A3B8685ULL, 0x4CD09FA50CAFD76CULL, 0x54CAA642B7132557ULL, 0x5CC34EE0218774BEULL,
0xA42F1AB01D8D50F9ULL, 0xAC26F2128B190110ULL, 0xB43CCBF530A5F32BULL, 0xBC352357A631A2C2ULL,
0x8408B83A47DC175DULL, 0x8C015098D14846B4ULL, 0x941B697F6AF4B48FULL, 0x9C1281DDFC60E566ULL,
0xE4605FA4A92FDFB1ULL, 0xEC69B7063FBB8E58ULL, 0xF4738EE184077C63ULL, 0xFC7A664312932D8AULL,
0xC447FD2EF37E9815ULL, 0xCC4E158C65EAC9FCULL, 0xD4542C6BDE563BC7ULL, 0xDC5DC4C948C26A2EULL,
0x49632132E9909CD2ULL, 0x416AC9907F04CD3BULL, 0x5970F077C4B83F00ULL, 0x517918D5522C6EE9ULL,
0x694483B8B3C1DB76ULL, 0x614D6B1A25558A9FULL, 0x795752FD9EE978A4ULL, 0x715EBA5F087D294DULL,
0x092C64265D32139AULL, 0x01258C84CBA64273ULL, 0x193FB563701AB048ULL, 0x11365DC1E68EE1A1ULL,
0x290BC6AC0763543EULL, 0x21022E0E91F705D7ULL, 0x391817E92A4BF7ECULL, 0x3111FF4BBCDFA605ULL,
0xC9FDAB1B80D58242ULL, 0xC1F443B91641D3ABULL, 0xD9EE7A5EADFD2190ULL, 0xD1E792FC3B697079ULL,
0xE9DA0991DA84C5E6ULL, 0xE1D3E1334C10940FULL, 0xF9C9D8D4F7AC6634ULL, 0xF1C03076613837DDULL,
0x89B2EE0F34770D0AULL, 0x81BB06ADA2E35CE3ULL, 0x99A13F4A195FAED8ULL, 0x91A8D7E88FCBFF31ULL,
0xA9954C856E264AAEULL, 0xA19CA427F8B21B47ULL, 0xB9869DC0430EE97CULL, 0xB18F7562D59AB895ULL,
0x6DD2B1AB9D58D2BBULL, 0x65DB59090BCC8352ULL, 0x7DC160EEB0707169ULL, 0x75C8884C26E42080ULL,
0x4DF51321C709951FULL, 0x45FCFB83519DC4F6ULL, 0x5DE6C264EA2136CDULL, 0x55EF2AC67CB56724ULL,
0x2D9DF4BF29FA5DF3ULL, 0x25941C1DBF6E0C1AULL, 0x3D8E25FA04D2FE21ULL, 0x3587CD589246AFC8ULL,
0x0DBA563573AB1A57ULL, 0x05B3BE97E53F4BBEULL, 0x1DA987705E83B985ULL, 0x15A06FD2C817E86CULL,
0xED4C3B82F41DCC2BULL, 0xE545D32062899DC2ULL, 0xFD5FEAC7D9356FF9ULL, 0xF55602654FA13E10ULL,
0xCD6B9908AE4C8B8FULL, 0xC56271AA38D8DA66ULL, 0xDD78484D8364285DULL, 0xD571A0EF15F079B4ULL,
0xAD037E9640BF4363ULL, 0xA50A9634D62B128AULL, 0xBD10AFD36D97E0B1ULL, 0xB5194771FB03B158ULL,
0x8D24DC1C1AEE04C7ULL, 0x852D34BE8C7A552EULL, 0x9D370D5937C6A715ULL, 0x953EE5FBA152F6FCULL,
0x92C64265D32139A4ULL, 0x9ACFAAC745B5684DULL, 0x82D59320FE099A76ULL, 0x8ADC7B82689DCB9FULL,
0xB2E1E0EF89707E00ULL, 0xBAE8084D1FE42FE9ULL, 0xA2F231AAA458DDD2ULL, 0xAAFBD90832CC8C3BULL,
0xD28907716783B6ECULL, 0xDA80EFD3F117E705ULL, 0xC29AD6344AAB153EULL, 0xCA933E96DC3F44D7ULL,
0xF2AEA5FB3DD2F148ULL, 0xFAA74D59AB46A0A1ULL, 0xE2BD74BE10FA529AULL, 0xEAB49C1C866E0373ULL,
0x1258C84CBA642734ULL, 0x1A5120EE2CF076DDULL, 0x024B1909974C84E6ULL, 0x0A42F1AB01D8D50FULL,
0x327F6AC6E0356090ULL, 0x3A76826476A13179ULL, 0x226CBB83CD1DC342ULL, 0x2A6553215B8992ABULL,
0x52178D580EC6A87CULL, 0x5A1E65FA9852F995ULL, 0x42045C1D23EE0BAEULL, 0x4A0DB4BFB57A5A47ULL,
0x72302FD25497EFD8ULL, 0x7A39C770C203BE31ULL, 0x6223FE9779BF4C0AULL, 0x6A2A1635EF2B1DE3ULL,
0xB677D2FCA7E977CDULL, 0xBE7E3A5E317D2624ULL, 0xA66403B98AC1D41FULL, 0xAE6DEB1B1C5585F6ULL,
0x96507076FDB83069ULL, 0x9E5998D46B2C6180ULL, 0x8643A133D09093BBULL, 0x8E4A49914604C252ULL,
0xF63897E8134BF885ULL, 0xFE317F4A85DFA96CULL, 0xE62B46AD3E635B57ULL, 0xEE22AE0FA8F70ABEULL,
0xD61F3562491ABF21ULL, 0xDE16DDC0DF8EEEC8ULL, 0xC60CE42764321CF3ULL, 0xCE050C85F2A64D1AULL,
0x36E958D5CEAC695DULL, 0x3EE0B077583838B4ULL, 0x26FA8990E384CA8FULL, 0x2EF3613275109B66ULL,
0x16CEFA5F94FD2EF9ULL, 0x1EC712FD02697F10ULL, 0x06DD2B1AB9D58D2BULL, 0x0ED4C3B82F41DCC2ULL,
0x76A61DC17A0EE615ULL, 0x7EAFF563EC9AB7FCULL, 0x66B5CC84572645C7ULL, 0x6EBC2426C1B2142EULL,
0x5681BF4B205FA1B1ULL, 0x5E8857E9B6CBF058ULL, 0x46926E0E0D770263ULL, 0x4E9B86AC9BE3538AULL,
0xDBA563573AB1A576ULL, 0xD3AC8BF5AC25F49FULL, 0xCBB6B212179906A4ULL, 0xC3BF5AB0810D574DULL,
0xFB82C1DD60E0E2D2ULL, 0xF38B297FF674B33BULL, 0xEB9110984DC84100ULL, 0xE398F83ADB5C10E9ULL,
0x9BEA26438E132A3EULL, 0x93E3CEE118877BD7ULL, 0x8BF9F706A33B89ECULL, 0x83F01FA435AFD805ULL,
0xBBCD84C9D4426D9AULL, 0xB3C46C6B42D63C73ULL, 0xABDE558CF96ACE48ULL, 0xA3D7BD2E6FFE9FA1ULL,
0x5B3BE97E53F4BBE6ULL, 0x533201DCC560EA0FULL, 0x4B28383B7EDC1834ULL, 0x4321D099E84849DDULL,
0x7B1C4BF409A5FC42ULL, 0x7315A3569F31ADABULL, 0x6B0F9AB1248D5F90ULL, 0x63067213B2190E79ULL,
0x1B74AC6AE75634AEULL, 0x137D44C871C26547ULL, 0x0B677D2FCA7E977CULL, 0x036E958D5CEAC695ULL,
0x3B530EE0BD07730AULL, 0x335AE6422B9322E3ULL, 0x2B40DFA5902FD0D8ULL, 0x2349370706BB8131ULL,
0xFF14F3CE4E79EB1FULL, 0xF71D1B6CD8EDBAF6ULL, 0xEF07228B635148CDULL, 0xE70ECA29F5C51924ULL,
0xDF3351441428ACBBULL, 0xD73AB9E682BCFD52ULL, 0xCF20800139000F69ULL, 0xC72968A3AF945E80ULL,
0xBF5BB6DAFADB6457ULL, 0xB7525E786C4F35BEULL, 0xAF48679FD7F3C785ULL, 0xA7418F3D4167966CULL,
0x9F7C1450A08A23F3ULL, 0x9775FCF2361E721AULL, 0x8F6FC5158DA28021ULL, 0x87662DB71B36D1C8ULL,
0x7F8A79E7273CF58FULL, 0x77839145B1A8A466ULL, 0x6F99A8A20A14565DULL, 0x679040009C8007B4ULL,
0x5FADDB6D7D6DB22BULL, 0x57A433CFEBF9E3C2ULL, 0x4FBE0A28504511F9ULL, 0x47B7E28AC6D14010ULL,
0x3FC53CF3939E7AC7ULL, 0x37CCD451050A2B2EULL, 0x2FD6EDB6BEB6D915ULL, 0x27DF0514282288FCULL,
0x1FE29E79C9CF3D63ULL, 0x17EB76DB5F5B6C8AULL, 0x0FF14F3CE4E79EB1ULL, 0x07F8A79E7273CF58ULL
};

uint64 Crc64(const unsigned char *pcBlock, unsigned int length)
{
        uint64 crc = 0xFFFFFFFFFFFFFFFFULL;
        while (length--)
            crc = (crc >> 8) ^ Crc64Table[(crc ^ *pcBlock++) & 0xFF];
        return crc;
}
```