Протокол бинарного формата данных ПРО

Трекер обменивается с сервером пакетами следующей структуры (бинарные контейнеры):

Заголовок пакета тип t_binary_container
Содержимое пакета - произвольные бинарные данные

В ответ сервер должен прислать трекеру пакет такой же структуры и произвольным содержимом. В случае если трекер получает корректный пакет с данной структурой в ответ на свою посылку пакет считается переданным успешно и трекер переходит к отправке нового пакета. В случае если ответного пакета не получено или получен некорректный пакет (не совпадает контрольная сумма или преамбула), исходный пакет считается отправленным с ошибкой и его отправка повторяется.

Описание типов данных:
uint8_t - 8ми битный целый беззнаковый.
int8_t - 8ми битный целый знаковый.

uint16_t - 16ти битный целый беззнаковый. Порядок записи байтов - обратный (от младшего к старшему, интеловский, как в x86)
int16_t - 16ти битный целый знаковый.

uint32_t - 32х битный целый беззнаковый.
int32_t - 32х битный целый знаковый.

Все пакеты упакованы. Т.е. выравнивание элементов структуры - 1 байт.

Алгоритм расчета контрольной суммы:

```
 1 /*
 2   Name  : CRC-16 CCITT
 3   Poly  : 0x1021    x^16 + x^12 + x^5 + 1
 4   Init  : 0xFFFF
 5   Revert: false
 6   XorOut: 0x0000
 7   Check : 0x29B1 ("123456789")
 8   MaxLen: 4095 (32767 bit)
 9 */
10
11 uint16_t crc16(unsigned char *pcBlock, uint16_t len)
```

```
12 {
13     unsigned short crc = 0xFFFF;
14     unsigned char i;
15
16     while(len--)
17     {
18         crc ^= *pcBlock++ << 8;
19
20         for(i = 0; i < 8; i++)
21             crc = crc & 0x8000 ? (crc << 1) ^ 0x1021 : crc << 1;
22     }
23
24     return crc;
25 }
```

Описание структуры **t_binary_container**:

```
1 typedef struct {
2   uint16_t crc;          // crc: sizeof (t_binary_container +
pay_load)
3   uint16_t preamble;
4   uint32_t tracker_id;
5   uint16_t data_len;
6 } t_binary_container;
```

crc - контрольная сумма пакета целиком начиная с поля preamble и заканчивая последним байтом содердимого пакета (кроме поля crc)
preamble - преамбула. Содердимое всегда 0x8A2C;
tracker_id - идентификатор трекера;
data_len - длина поля с данными;

Максимальная длина бинарного контейнера - 1400байт, включая заголовок.

В качестве полезного содержимого бинарного контейнера выступают данные со структурой **t_common_data_header** за которой следуют полезные данные пакета:

Заголовок - t_common_data_header

Полезные данные пакета

Таких пакетов в контейнере может быть несколько (а может не быть вовсе).

Описание структуры **t_common_data_header**:

```
typedef struct {
  uint16_t crc;
  uint16_t serial_id;
  uint32_t timestamp;
  uint16_t packet_type;
  uint16_t packet_len;
  uint32_t x_coord;
  uint32_t y_coord;
  uint8_t sf;
} t_common_data_header;
```

crc - контрольная сумма пакета, включая блок с данными (кроме поля crc);
serial_id - серийный номер пакета
timestamp - unixtime пакета (UTC+0)
packet_type - тип структуры, которая является полезными данными текущего пакета;
packet_len - длина полезных данных;
x_coord - широта - в соответсвии с правилами ЕГТС. Если равно 0xffffffff - значит нет фиксации валидных координат;
y_coord - долгота - в соответствии с правилапи ЕГТС. Если равно 0xffffffff - значит нет фиксации валидных координат;
sf - Дополнительные флаги пакета - битовая маска:

бит 0 - признак отправки данных из черного ящика (0 - свежесобранные данные, 1 - данные отправлены из черного ящика);
бит 6 - поле LAHS записи EGTS_SR_POS_DATA; 0 - северная широта; 1 - южная широта;
бит 7 - поле LOHS записи EGTS_SR_POS_DATA; 0 - восточная долгота; 1 - западная долгота;

Описание типов пакетов:

B_PACKET_TYPE_RCPTOK_SIMPLE = 0xFFFF - пустой пакет (для подтверждения приема контйнера). Данных нет, поле packet_len=0

B_PACKET_TYPE_SENSORS_V1 = 0x0001 - содержит в себе данные датчиков трекера:

```
typedef struct {
  uint16_t update_reason;
```

```
 3   uint16_t d_state;
 4   uint16_t v_bat;
 5   uint16_t v_in;
 6   uint16_t v_5v;
 7   uint16_t v_1224v;
 8   uint16_t pwr_flags;
 9   uint16_t acc_x;
10   uint16_t acc_y;
11   uint16_t acc_z;
12   uint8_t  adc_chan_to_send;
13   uint8_t  discreete_count_to_send;
14   uint8_t  ow_count_to_send;
15   uint8_t  ibutton_data_send;
16   t_adc_data adc_data[adc_chan_to_send];
17   t_discreete_data discreete_data[discreete_count_to_send];
18   t_ow_data    ow_data[ow_count_to_send];
19   t_ibutton_data ibutton_data[ibutton_data_send];
20 } t_sensor_data;
```

update_reason - причина отправки данного сообщения - бинарное поле:

SE_ADC_CHANGED 0x0001 - изменения показаний аналоговых датчиков выше критического значения
SE_FREQ_CHANGED 0x0002 - изменения показаний частотных датчиков выше критического значения
SE_ACC_CRITICAL_EVENT 0x0004 - изменения показаний акселерометра выше критического значения
SE_IBUTTON_EVENT 0x0008 - приложена кнопка ibutton
SE_DS1820_EVENT 0x0010 - изменения показаний температурных датчиков выше критического значения
SE_TIMER_EVENT 0x8000 - обновление данных по таймеру.

Причин обновлений может быт несколько. В этом случае они комбинируются по OR;

d_state - текущее состояние дискретных входов

бит 0: состояние цифрового входа 1 (1 - активен; 0 - не активен);
бит 1: состояние цифрового входа 2 (1 - активен; 0 - не активен);
бит 2: состояние цифрового входа 3 (1 - активен; 0 - не активен);
бит 3: состояние цифрового входа 4 (1 - активен; 0 - не активен);

бит 4: состояние цифрового входа 5 (1 - активен; 0 - не активен);
бит 5: состояние цифрового входа 6 (1 - активен; 0 - не активен);
бит 6: зарезервирован;
бит 7: зарезервирован;

v_bat - напряжение на аккумуляторе - сотые доли вольта;
v_in - входное напряжение на трекере - сотые доли вольта;
v_5v - напряжение по цепи 5в в трекере - сотые доли вольта;
v_1224v - напряжение по цепи 12-24в - сотые доли вольта;
pwr_flags - текущее состояние трекера:

#define POWER_MODE_SLEEP 0x01 - трекер находится в спящем режиме
#define POWER_MODE_ACTIVE 0x02 - трекер находится в активном режиме
#define MODE_ACC_STOP 0x04 - зафиксирована остановка по акселерометру
#define IGNITION_ON 0x08 - включено зажигание
#define DEBUG_OUTPUT 0x10 - включен вывод отладочной информации в консоль
#define SENSOR_OUTPUT 0x20 - включен вывод значения датчиков в консоль
#define CAN_DEBUG_OUTPUT 0x40 - включен вывод CAN
#define NMEA_OUTPUT 0x80 - включен вывод NMEA

acc_x - ускорение по оси X - от 0 до 32767. В диапазоне работы акселерометра, который задается переменной ACC_RANGE
acc_y - ускорение по оси Y - от 0 до 32767.
acc_z - ускорение по оси Z - от 0 до 32767.
adc_chan_to_send - количество каналов АЦП к отправке - от 0 до 2
discreete_count_to_send - количество дискретных каналов к отправке - от 0 до 6
ow_count_to_send - количество каналов термометров к отправке - от 0 до 16
ibutton_data_send - количество каналов ibutton к отправке - 0 или 1
adc_data[N1] - данные АЦП. Количество полей зависит от поля adc_chan_to_send. Может отсутсвовать.
discreete_data[N2] - данные дискретных датчиков. Количество полей зависит от поля discreete_count_to_send. Может отсутствовать.
ow_data[N3] - данные температурных датчиков. Количество полей зависит от поля ow_count_to_send. Может отсутствовать.
ibutton_data[N4] - данные кнопки ibutton. Может отсутствовать

Описание структур:

```
 1 typedef struct {

 2   uint16_t adc_data; // напряжение на соответсвующем канале АЦП - 
сотые доли вольта

 3 } t_adc_data;

 4

 5 typedef struct {

 6   int32_t freq; // частота на соответсвующем канале дискретных 
датчиков

 7   uint32_t counter; // состояние (счетчик переключений) 
соответсвующего дискретного датчика;
```

```
 8 } t_discreete_data;

 9

10 typedef struct {

11   int16_t ow_data; // температура соответсвующего канала
температурных датчиков, целое со знаком, СОТЫЕ ДОЛИ ГРАДУСА

12 } t_ow_data;

13

14 typedef struct {

15   uint8_t  i_button_id[8];  // ID ключа

16   uint32_t i_button_fix_time; // точное время "приложения" ключа.
unixtime UTC+0.

17 } t_ibutton_data;
```

B_PACKET_TYPE_GSM_V1 0x0002 - содержит в себе данные по GSM каналу

```
 1 typedef struct {

 2   uint8_t module_temperature;  // температура GSM модуля

 3   uint8_t gsm_strength;        // уровень GSM - сигнала 0..30ед

 4   uint8_t network_register_state; // Состояние регистрации в сети. 0
- нет регистрации, 1 - домашняя сеть, 5 - роуминг;

 5   uint8_t active_sim_number; // номер активной СИМ-карты

 6   uint8_t active_server_number; // Номер активного сервера

 7   char operator_name[8]; // Название активного сотового оператора
(часть названия)

 8   uint16_t operator_id; // ID сотового оператора

 9   uint16_t mcc; // Данные LBS - MCC

10   char lac[5]; // Данные LBS - LAC

11   uint16_t mnc; // Данные LBS - MNC

12   uint16_t rxl;  // Данные LBS - RXL

13   uint16_t rxq; // Данные LBS - RXQ

14   uint16_t ta;  // Данные LBS - TA

15   uint16_t bsic;  // Данные LBS - BSIC

16   char cell_id[5]; // Данные LBS - CellID в HEX (!!!)

17   uint16_t mdm_cmd_flag; // заразервировано

18   uint8_t max_idle_counter; // зарезервировано
```

```
19   uint8_t gprs_state; // состояние GPRS подключения. 0 - не активно,
1 - активно;

20   uint8_t tcp_state; // состояние TCP соединения. 0 - не активно. 1 -
активно;

21   char imei[20]; // IMEI

22   char iccid[21]; // ICCID

23   uint8_t egts_last_error; // Состояние последнего обращения к
серверу ЕГТС (см расшифровку ниже)

24   uint8_t egts_error_param_1; // Параметр ошибки сервера ЕГТС №1
(если он есть)

25   uint8_t egts_error_param_2; // Параметр ошибки сервера ЕГТС №2
(если он есть)

26 } t_gsm_info;
```

B_PACKET_TYPE_AV_V1 0x0003 - данные GPS

```
 1 typedef struct {

 2   uint16_t v_in;             // входное напряжение, сотые доли вольта

 3   uint16_t v_bat;            // напряжение на акумуляторе, сотые доли
вольта

 4   int16_t fs_data;           // данные топливного датчика

 5   uint8_t  mode_acc_stop:4;  // состояние остановки по акселерометру.
0 - остановк ане зафиксирована; 1 - зафиксирована остановка;

 6   uint8_t  ignition_on:4;    // состояние датчика зажигания. 0 -
зажигание выключено; 1 - зажигание включено;

 7   uint8_t  d_state;          // состояние дискретныз датчиков - см.
выше

 8   uint8_t  fix_type;         // тип фиксации GPS/GLONASS;

 9   uint8_t  sat_count;        // количество отслеживаемых спутников

10   uint16_t  hdop;            // hdop - сотые доли

11   uint16_t altitude;         // Высота антенны приёмника над/ниже
уровня моря, м;

12   uint16_t geoid_height;     // Геоидальное различие - различие между
земным эллипсоидом WGS-84 и уровнем моря(геоидом), ”-” = уровень моря
ниже эллипсоида.;

13   uint16_t speed;            // скорость - сотые доли км-ч

14   uint16_t azimuth;          // вектор перемещения - сотые доли
градусов

15   int32_t  x_coord;          // широта - кодирование в соответсвии с
правилами ЕГТС; (4294967295 - координата невалидна)
```

```
16   int32_t  y_coord;          // долгота - кодирование в соответсвии с
правилами ЕГТС; (4294967295 - координата невалидна)

17   uint8_t ant_state;         // состояние GPS антенны. 0 - OK, 1 -
короткое замыкание, 2 - отсутсвует или обрыв.

18   uint8_t  egts_flags;       // флаги ЕГТС для передачи на сервер

19   uint8_t  egts_src;         // Источник данных ЕГТС для передачи на
сервер

20 } t_gps_data_v4;
```

B_PACKET_TYPE_DUT_E 0x0004 - данные топливного датчика

```
 1 typedef struct {

 2   uint8_t  bind_place;  // место подключения датчика (0 - RS232_MAIN;
1 - RS232_AUX; 2 - RS485);

 3   char     t;           // температура топлива;

 4   uint16_t n;           // текущий уровень топлива;

 5   uint16_t f_curr;      // текущая частота внутреннего генератора;

 6   uint32_t id;          // ID датчика;

 7   uint16_t f_max_t;     // показания частоты при максимальном уровне
топлива;

 8   uint16_t f_min_t;     // показания частоты при минимальном уровне
топлива;

 9   uint16_t k_t;         // зарезервировано;

10   char     k_t0;        // зарезервировано;

11   uint16_t pwm_max;     // зарезервировано;

12   uint16_t trl;         // зарезервировано;

13   char net_addr;        // сетевой адрес датчика;

14   char net_mode;        // режим работы датчика - 0 - одиночный; 1 -
сетевой;

15   char pwm_mode;        // зарезервировано

16 } t_dut_e_struct;
```

B_PACKET_TYPE_CAN_V1 0x0005 - данные с CAN шины

Важно !!! Пакет содержит 4 пакета **t_can_data**

```
1 typedef struct {

2   uint32_t timestamp; // время захвата пакета - unixtime

3   uint32_t can_id;    // can_id пакета
```

```
4   uint8_t  can_data[9];  // данные can-пакета (can_data[0] - длина
пакета; can_data[1]...can_data[8] - данные пакета)

5 } t_can_data;
```

PACKET_TYPE_ALARM_DATA 0x000A - данные тревожной кнопки

```
1 typedef struct {

2   uint16_t gps_speed;             // Скорость в момент возникновения
тревожного события

3   uint16_t gps_angle;             // Направление в момент
возникновения тревожного события

4   uint32_t x_coord;               // Широта - милионные доли градуса

5   uint32_t y_coord;               // Долгота - милионные доли градуса

6   uint8_t  alarm_state;           // 1 - есть тревога; 0 - нет тревоги

7 } t_alarm_data_v1;
```

PACKET_TYPE_FW_ASK 0x0006 - запрос блока прошивки

```
1 typedef struct {

2   unsigned char cur_fw_version[12]; - текущая версия прошивки в
устройстве

3   unsigned char req_fw_version[12]; - запрашиваемая версия прошивки.
Если 0x00, то запрашивается последняя доступная версия

4   unsigned char serial[12];         - серийный номер устройства

5   unsigned char device_type;        - тип устройства

6   uint32_t      fw_flash_ptr;       -  смещение с которого
запрашивается прошивки

7 } t_fw_ask_data;
```

PACKET_TYPE_FW_PAGE_DATA 0x0007 - блок прошивки

```
1 #define FW_PAGE_DATA_SIZE  1028

2 typedef struct {

3   uint32_t offset;                - смещение блока

4   uint32_t fw_size;               - общий размер прошивки

5   unsigned char fw_version[12];   - весия отдаваемой прошивки

6   unsigned char device_type;      - для какого устройсва
предназначена прошивка

7   uint8_t  fw_page_data[FW_PAGE_DATA_SIZE]; - блок прошивки
```

```
8 } t_fw_page_data;
```

PACKET_TYPE_TEXT_CMD_DATA - 0x0008 - текстовая команда устройству из стека команд

```
1 typedef struct {
2   char     cmd_id[16]; - ID команды
3   char     cmd[160];   - Команда устройству
4 } t_text_cmd_data;
```

При получении этой команды терминал выполняет команду и этим же пакетом пересылает результат ее выполнения

PACKET_TYPE_CONFIG_POLL - 0x0009 - запрос данных с конфигурационного сервера

```
1 typedef struct {
2   uint8_t serial[12]; - серийный номер устройства
3 } t_poll_data;
```

Трекер запрашивает данные с конфигурационного сервера. В ответ терминал ожидает структуру PACKET_TYPE_TEXT_CMD_DATA в составе пакета или пустой пакет если команд этому терминалу нет

PACKET_TYPE_DEBUG - отладочная информация

```
 1 typedef struct {
 2   uint32_t last_reset_state; - причина последней перезагрузки
 3   uint32_t flash_rd;         - указатель памяти на чтение
 4   uint32_t flash_wr;         - указатель памяти на запись
 5   uint32_t uptime_cnt;       - счетчик uptime
 6   t_gps_stat gps_stat;       - статистика по работе GPS модуля
 7 } t_debug_packet;
 8
 9 typedef struct {
10   uint32_t crc_err_cnt; //+    L
11   uint32_t no_data_cnt;    //+ L
12   uint32_t not_processed_cnt; //+  L
13   uint8_t gps_power_state;   //+   C
```

```
14   uint32_t gps_power_cycles_counter;//+  L
15   uint32_t angle_int; //+                L
16   uint32_t speed_int; //+                L
17   uint8_t gps_ok;     //+                C
18   uint8_t stop_state; //+                C
19   uint8_t sleep_state; //+               C
20   uint8_t freeze_acc; //+                C
21   uint8_t freeze_ign; //+                C
22   uint8_t sat_count;  //+                C
23   uint8_t fix_type;   //+                C
24   uint32_t send_timer;    //+            L
25   uint32_t send_interval; //+            L
26   uint16_t speed_change;  //+            S
27   uint16_t angle_change;  //+            S
28   uint8_t last_gps_event; //+            C
29   uint8_t has_gps_fix; //+               C
30 } t_gps_stat;
31
```

PACKET_TYPE_CAN_LOG_DATA - 0x000D - данные с CANLog (дочерней платы расширения или модуля, подключаемого по интерфейсу RS232/485);

```
1 typedef struct {
2   t_can_log_frame can_log_frames[N];
3 } t_can_log_data;
4
5 typedef struct {
6   char     prefix[2]; // S,
A,B,C,D,E,F,G,R,H,I,J,K,L,M,N,O,P,U,V,WA,WB,WC,WD,WE,WF,WG,WH,TA,TB,TC,T
D,TE,XA,XB,Z
7   uint32_t data;
8 } t_can_log_frame;
```

Пакет t_can_log_data состоит из нескольких пакетов типа t_can_log_frame

Точное число пакетов в сообщении рассчитывается по формуле t_common_data_header.packet_len/sizeof(t_can_log_frame). При этом максимальное количество пакетов t_can_log_frame ограничего 40;

Интерпретация пакета:

поле prefix определяет содержимое пакета. Поле может состоять из одного или двух символов. В случае если символ один, второй символ - 0x00.
Поле data - данные, соответствующие содержимому

prefix = S - security state flags. data - security state flags (битовое поле) - описание см в файле в аттаче;
A или B - полное время работы двигателя. data - сотые доли часа;
C или D - полный пробег транспортного средства. data - пробег ТС в метрах;
E или F - полный расход топлива. data - расход топлива в десятых долях литра;
G - уровень топлива в баке в %. data - уровень топлива в баке в десятых долях процента;
R - уровень топлива в баке в литрах. data - уровень топлива в баке в десятых долях литра;
H - обороты двигателя в минуту (rpm). data - скорость вращения коленвала двигателя (об/мин);
I - температура двигателя (град С). data - температура двигателя - град С;
K - нагрузка на ось 1. data - нагрузка на ось 1 в десятых долях килограмма;
L - нагрузка на ось 1. data - нагрузка на ось 1 в десятых долях килограмма;
M - нагрузка на ось 1. data - нагрузка на ось 1 в десятых долях килограмма;
N - нагрузка на ось 1. data - нагрузка на ось 1 в десятых долях килограмма;
O - нагрузка на ось 1. data - нагрузка на ось 1 в десятых долях килограмма;
P - контроллеры аварии. data - контролеры аварии (битовое поле) - описание см в файле в аттаче;
U - уровень жидкости adBlue в баке в %. data - уровень жидкости abBlue в баке в десятых долях процента;
V - уровень жидкости adBlue в баке в литрах. data - уровень жидкости adBlue в баке в десятых долях литра;
WA - состояние сельхозтехники. data - состояние сельхозтехники (битовая маска)
WB - время жатки. data - время жатки, сотые доли часа
WC - убранная площадь. data - убранная площадь, сотые доли га
WD - производительность. data - производительность, сотые доли га в час
WE - количество собранного уражая. data - количество собранного урожая, сотые доли тонны
WF - влажность зерна. data - влажность зерна, десятые доли %
WG - обороты молотильного барабана. data - обороты молотильного барабана, rpm (обороты в минуту)
WH - зазор подбарабанья на выходе. data - зазор подбарабанья на выходе, мм
TA,TB,TC,TD,TE - информация с этих кадров CANLog не обрабатывается
XA - положение педали газа. data - положение педали газа, %
XB - нагрузка на двигатель. data - нагрузка на двигатель, %

B_PACKET_TYPE_TEXT_DATA - 0x000F - Содержит в ASCIIZ строку, принятую с 232/485го порта.

```
1 typedef struct {

2   char data[N];

3 } t_str_data;
```

Длинна массива передается в заголовке t_common_data_header.

PACKET_TYPE_FMS_DATA - 0x0011 - данные FMS (с CAN-шины);

```
typedef struct {
  t_fms_field fms_frames[N];
} t_fms_data;

typedef struct {
  char     field_name[12];
  uint32_t field_value;
} t_fms_field;
```

Пакет t_fms_data состоит из нескольких пакетов типа t_fms_field

Точное число пакетов в сообщении рассчитывается по формуле t_common_data_header.packet_len/sizeof(t_fms_field). При этом максимальное количество пакетов t_fms_field ограничено 40;

Интерпретация пакета:

поле field_name (ASCIIZ строка) определяет содержимое пакета.
Поле data - данные, соответствующие содержимому

Описание полей:

fuel_lfc - fuel consumption, ltr
fuel_prc - fuel level, %
torq_prc - Engine torque, %
rpm - Engine speed, rpm
hours - Engine total hours or operation, hours
milage - High resolution total milage, m
engine_tmp - Engine temperature, C
amb_tmp - Abmient temperature, C
fuel_rate - Fuel rate l/hour
fuel_eco - Fuel economy km/l
air_pres - Supply air pressure, kpa
fuel_hrlfc - Total fuel used, l
adblue_prc - Aftertreatment 1 Diesel Exhaust Fluid Tank 1 level, %
fms1_1 - FMS Tell Tale Status: FMS1 P1-P4
fms1_2 - FMS Tell Tale Status: FMS1 P5-P8
prk_brk - Parking brake state
speed - Wheel based speed, km/h
pedal_state - b8-7: clutch switch/b6-5: brake switch/b2-1: cruise control state
pto_state - PTO state
acc_pos - Accelerator pedal position, %

engine_load - Engine load, %
axle_tire_l - Axle/tire location
axle_wght - Axle weight, kg
srv_dst - Service distance, km
pto_de - PTO Drive Engagement: PTODE
gross_wght - Gross weight, kg
rtrd_mode - Retarder mode
rtrd_torq - Retarder torque, %
rtrd_sel - Retarder selection non-engine, %
door_st1 - Doors state 1
door_st2 - Doors state 2
door_st3 - Doors state 3
fms_time - FMS time
fms_date - FMS date
alt_state - Alternator state
sel_gear - Selected gear
cur_gear - Current gear
b_pres_fl - Bellow pressure front left, kpa
b_pres_fr - Bellow pressure front right, kpa
b_pres_rl - Bellow pressure rear left, kpa
b_pres_rr - Bellow pressure rear right, kpa

Номера пакетов и их идентификаторы:

#define B_PACKET_TYPE_SENSORS_V1 0x0001
#define B_PACKET_TYPE_GSM_V1 0x0002
#define B_PACKET_TYPE_AV_V1 0x0003
#define B_PACKET_TYPE_DUT_E 0x0004
#define B_PACKET_TYPE_CAN_V1 0x0005
#define B_PACKET_TYPE_FW_ASK_V1 0x0006
#define B_PACKET_FW_PAGE_DATA_V1 0x0007
#define B_PACKET_TYPE_TEXT_CMD_DATA 0x0008
#define B_PACKET_TYPE_CONFIG_POLL 0x0009
#define B_PACKET_TYPE_ALARM_DATA 0x000A
#define B_PACKET_TYPE_DEBUG 0x000B
#define B_PACKET_TYPE_IMAGE_DATA 0x000C
#define B_PACKET_TYPE_CAN_LOG_DATA 0x000D
#define B_PACKET_TYPE_TEXT_DATA 0x000F

Пример пакта с данными:

t_binary_container
-- t_common_data_header
---- t_gps_data_v4
-- t_common_data_header
---- t_sensor_data

-- t_common_data_header
---- t_gsm_info

Протокол бинарного формата данных Лайт

Данный протокол является дополнением к бинарному формату GPS.PRO. Описаны только пакеты, отсутствующие в версии ПРО.

Тип пакета 0x0040 - информация о GPS-координатах

```
 1 typedef struct {

 2   uint16_t v1224; - напряжение на аккумуляторе автомобиля, сотые доли
вольта

 3   uint16_t v_bat; - напряжение на аккумуляторе трекера - сотые доли
вольта;

 4   uint16_t fs_data; - данные топливного датчика 1 на канале RS232 или
RS485

 5   uint8_t  stop_state; - 1 если автомобиль стоит на месте; 0 - если
едет;

 6   uint8_t  ign_state; - 1 если зажигание включено; 0 - если зажигание
выключено;

 7   uint8_t  d_state; - состояние дискретных входов - битовая маска;

 8   uint16_t freq2; - частота на дискретном входе 2;

 9   uint16_t c_counter2; - счетчик импульсов на дискретном входе 2;

10   uint16_t freq1; - частота на дискретном входе 1;

11   uint16_t c_counter1; - счетчик импульсов на дискретном входе 1;

12   uint8_t  ant_state; - состояние GPS-антенны;

13   uint8_t  fix_type; - тип фиксации спутников;

14   uint8_t  sat_count; - количество отслеживаемых спутников;

15   uint16_t altitude; - высота;

16   uint16_t geoid_height; - высота геоида;

17   uint32_t x_coord; - широта в системе шифрования ЕГТС;

18   uint32_t y_coord; - долгота в системе шифрования ЕГТС;

19   uint16_t speed; - сотые км-час;

20   uint16_t course; - азимут, градусы;

21   uint16_t adc1; - данные АЦП1 - сотые доли вольта;

22   uint16_t c_counter3; - счетчик дискретного входа 3;

23   int16_t  ow_data; - данные датчика 1-wire N1, сотые доли градуса;

24   uint8_t  egts_flags; - флаги ЕГТС;
```

```
25   uint8_t  egts_src; - источник данных ЕГТС;

26 } t_l2b_gps_info;
```

Тип пакета 0x0041 - информация о GSM-сети

```
 1 typedef struct {

 2   char operator_name{8}; название оператора. ASCIIZ строка. Максимум
7 символов + 1 терминатор

 3   uint8_t active_sim_number; - номер активной СИМ-карты (1 или 2)

 4   uint8_t active_server_number; - номер активного сервера (1 или 2)

 5   uint8_t network_register_state; - состояние регистрации в GSM сети
(0,2 - нет регистрации 1 - регистрция в домашней сети; 5 - роуминг)

 6   uint8_t gsm_strength; - мощность GSM сигнала - от 0 до 31 ед

 7   uint8_t module_temperature; -  температура GSM модуля в градусах

 8   uint16_t operator_id; - ID сотового оператора

 9   char lac{5}; LAC - ASCIIZ строка

10   uint16_t mnc; - MNC

11   char cell_id{5}; - CELL_ID - ASCIIZ строка;

12   uint16_t mcc; - MCC

13   uint16_t bsic;- BSIC

14   uint16_t rxl; -  RXL

15   uint16_t rxq; - RXQ

16   uint16_t ta; - TA

17 } t_l2b_gsm_info;
```

Тип пакета 0x0042 - информация о датчиках

```
1 typedef struct {

2   uint16_t acc_max_x; - данные акселерометра по оси X

3   uint16_t acc_max_y; - данные акселерометра по оси Y

4   uint16_t acc_max_z; - данные акселерометра по оси Z

5   uint16_t fuel_data{4}; - данные 4х топливных датчиков RS485

6   uint32_t uptime_cnt; - время с момента запуска трекера, сек

7 } t_l2b_sd_line;
```

Тип пакета 0x0043 - отладочная информация

```
 1 typedef struct {

 2   uint32_t uptime_cnt; - время с момента запуска трекера, сек

 3   uint32_t wr_flash_address; - указатель адреса записи во flash
память

 4   uint32_t rd_flash_address; - указатель адреса чтения из flash-
памяти

 5   uint32_t rcc_csr; - флаги запуска трекера

 6   char imei{20}; - IMEI модема

 7   char iccid{21}; -  ICCID SIM карты

 8   uint16_t vcc; - напряжение питания по шине 3.3V, сотые доли вольта

 9 } t_l2b_debug_info_line;
```

Тип пакета 0x0044 - информация о датчиках 1-wire

```
1 typedef struct {

2   t_single_ow_data ow_data{4}; - данные 4х температурных датчоков

3   uint32_t i_button_fix_time; - время последнего считывания кнопки
iButton, unixtime

4   uint8_t i_button_id{8}; - ID кнопки - 8 байт

5 } t_l2b_ow_info_line;
```

Структура t_single_ow_data описана ниже:

```
1 typedef struct {

2   uint8_t ow_id; - идентификатор температурного датчика - последний
байт OW-ID датчика

3   int16_t ow_temp; - температура, сотые доли градуса

4 } t_single_ow_data;
```

Тип пакета 0x0045 - минимальная информация о маршруте

```
1 typedef struct {

2   uint8_t  ant_state;  - состояние GPS антенны

3   uint8_t  fix_type;   - тип фиксации спутников

4   uint8_t  sat_count;  - количество спутников в обзоре

5   uint16_t speed;      - текущая скорость - сотые км-ч

6   uint16_t course;     - текущий вектор перемещения

7   uint8_t egts_flags;  - флаги ЕГТС
```

```
8 } t_l2b_gps_info_min;
```

Тип пакета 0x004A - расширенная информация по GPS-модулю

```
1 typedef struct {

2   uint8_t sv_gps;       - количество видимых спутников GPS

3   uint8_t sv_glonass;   - количество видимых спутников GLONASS

4   uint8_t sc_gps;       - количество отслеживаемых спутников GPS

5   uint8_t sc_glonass;   - количество отслеживаемых спутников GLONASS

6   uint8_t snr_min;      - минимальное значение SNR отслеживаемых
спутников, dbHz

7   uint8_t snr_max;      - максимальное значение SNR отслеживаемых
спутников, dbHz

8 } t_l2b_gps_ex_info;
```

Тип пакета 0x004B - универсальный пакет GPS-Lite

```
 1

 2 typedef struct {

 3   uint32_t flags1;       // Битовое поле определяющее присутствие
полей группы 0 в пакете

 4   uint8_t  ant_state;    // поля от сюда

 5   uint8_t  fix_type;

 6   uint8_t  sat_count;

 7   uint16_t speed;

 8   uint16_t course;

 9   uint8_t egts_flags;   // до сюда присутствуют в пакете всегда

10   // Остальные поля присутствуют в пакете только если установлен
соответствующий бит в поле flags1 или flags2

11   uint32_t flags2;            // 0.0 - битовое поле определяющее
присутствие полей группы 1 в пакете

12   t_altitude_info altitude_info; // 0.1 - информация о высоте

13   uint16_t v1224;        // 0.2 - напряжение на входе питания
трекера, сотые вольта

14   uint16_t v_bat;        // 0.3 - напряжение на аккумуляторе трекера,
сотые вольта

15   uint16_t adc1;         // 0.4 - напряжение на аналоговом входе 1
трекера, сотые вольта
```

```
16
17   uint16_t freq1;        // 0.5 - частота, измеренная на частотном
входе 1, Гц
18   uint16_t freq2;        // 0.6 - частота, измеренная на частотном
входе 2, Гц
19   uint16_t c_counter1;   // 0.7 - количество импульсов, подсчитанных
на дискретном входе 1, шт
20   uint16_t c_counter2;   // 0.8 - количество импульсов, подсчитанных
на дискретном входе 2, шт
21   uint16_t c_counter3;   // 0.9 - количество импульсов, подсчитанных
на дискретном входе 3, шт
22
23   uint16_t fuel_data1;   // 0.10 - показания первого топливного
датчика на входе RS485/232
24   uint16_t fuel_data2;   // 0.11 - показания второго топливного
датчика на входе RS485
25   uint16_t fuel_data3;   // 0.12 - показания третьего топливного
датчика на входе RS485
26   uint16_t fuel_data4;   // 0.13 - показания четвертого топливного
датчика на входе RS485
27
28   uint8_t  stop_state;   // 0.14 - 1 - трекер находится в состоянии
покоя, 0 - трекер находится в движении
29   uint8_t  ign_state;    // 0.15 - 1 - зажигание включено, 0 -
зажигание выключено
30   uint8_t  d_state;      // 0.16 - состояние дискретных входов
(битовая маска)
31   uint8_t  egts_src;     // 0.17 - источник данных ЕГТС
32
33   uint16_t acc_data[3];  // 0.18, массив с показаниями акселерометра
по осям X,Y,Z
34
35   t_single_ow_data ow_data1; // 0.19 - номер и результаты замеров
температурного датчика 1 (см описание структуры t_single_ow_data ниже)
36   t_single_ow_data ow_data2; // 0.20 - номер и результаты замеров
температурного датчика 2
37   t_single_ow_data ow_data3; // 0.21 - номер и результаты замеров
температурного датчика 3
38   t_single_ow_data ow_data4; // 0.22 - номер и результаты замеров
температурного датчика 4
```

```
39

40   t_ibutton_data   ibut_data; // 0.23 - данные по кнопке iButton  (см описание структуры t_ibutton_data ниже)

41

42   char operator_name[8];                     // 0.24 - ASCIIZ строка - наименование текущего сотового оператора

43   uint8_t active_sim_number;                 // 0.25 - номер текущей сим-карты (0 - СИМ-карта номер 1, 1 - симкарта номер 2)

44   uint8_t active_server_number;              // 0.26 - номер текущего сервера (0/1)

45   uint8_t network_register_state;            // 0.27 - состояние регистрации в сети (0 - нет регистрации, 1 - регистрация в домашней сети, 5 - регистрация в роуминге)

46   uint8 t gsm strength;                      // 0.28 - мощность GSM сигнала (0 - минимум, 31 - максимум)

47   uint8_t module_temperature;                // 0.29 - температура GSM модуля

48   uint16_t operator_id;                      // 0.30 - operator_id

49   uint32_t uptime_cnt;                       // 0.31 - время работы трекера от последнего включения / перезагрузки, сек

50

51   char lac[5];                               //1.0 - lac (ASCIIZ)

52   uint16_t mnc;                              //1.1 - mnc

53   char cell_id[5];                           //1.2 - cell_id (ASCIIZ)

54   uint16_t mcc;                              //1.3 - mcc

55   uint16_t bsic;                             //1.4 - bsic

56   uint16_t rxl;                              //1.5 - rxl

57   uint16_t rxq;                              //1.6 - rzq

58   uint16_t ta;                               //1.7 - ta

59   uint32_t distance_travalled;               //1.8 - пробег ТС

60 } t_universal_data_full;

61

62 typedef struct {

63   int16_t altitude;

64   int16_t geoid_height;
```

```
65 } t_altitude_info;

66

67 typedef struct {

68   uint8_t ow_id;

69   int16_t ow_temp;

70 } t_single_ow_data;

71

72 typedef struct {

73   uint8_t  i_button_id[8];

74   uint32_t i_button_fix_time;

75 } t_ibutton_data;
```

Биты входящие в состав флагов flags1 и flags2 определяют наличие соответсвующих полей в структуре t_universal_data_full.
Например:
- бит 0 поля flags1 определяет присутствие поля 0.0 (flags2). Если flags2 отсутствует, то отсутствуют и все поля группы 1.* (lac, mnc, cell_id и т.д...).
- бит 1 поля flags1 определяет присутствие поля 0.1 (altitude_info);
- бит 2 поля flags1 определяет присутствие поля 0.2 (v1224);
- бит 3 поля flags1 определяет присутствие поля 0.3 (v_bat);

и т.д...

Тип пакета 0x004C - универсальный пакет GPS-Lite3:

Заголовок:

```
1 typedef struct {

2   uint8_t  ant_state;    // поля от сюда

3   uint8_t  fix_type;

4   uint8_t  sat_count;

5   uint16_t speed;

6   uint16_t course;

7   uint8_t  reason;   // до сюда присутсвуют в пакете всегда

8 } t_l4_data_base;
```

Далее за этим пакетом идет произвольный набор полей со следующей структурой:

тип поля 1 - 1 байт
данные поля 1 - длина зависит от типа поля
тип поля 2 - 1 байт
данные поля 2 - длина зависит от типа поля

Если тип поля = 255, то дальше данных нет - трекер не накопил данных что бы заполнить все поля полностью.

Таким образом каждый пакет может содержать в себе до 255 полей.

Тип и название полей пока статичны и определяются следующей структурой:

```
1 typedef struct {
2   uint8_t field_idx;
3   uint8_t field_len;
4   char    field_name[12];
5 } t_field_def;
6
```

В настоящее время определены следующие типы полей:

```
 #define FIELD_TYPE_8B     1
 #define FIELD_TYPE_16B    2
 #define FIELD_TYPE_32B    4
 #define FIELD_TYPE_FLOAT  4
 #define FIELD_TYPE_16Z    16
 #define FIELD_TYPE_20Z    20

 const t_field_def field_defs[]={
   //                    012345678901
0, FIELD_TYPE_16B,   "alt",   // *      //3 #alias:altitude
  1, FIELD_TYPE_16B,   "v_in",  // *      //3 #alias:innervoltage
  2, FIELD_TYPE_8B,    "ign_state", // *  //2 #alias:ign
  3, FIELD_TYPE_16B,   "vbat",  // *      //3 #alias:battery
  4, FIELD_TYPE_16B,   "adc1",  // *      //3
```

```
5, FIELD_TYPE_16B,   "adc2",  // *      //3
6, FIELD_TYPE_16B,   "freq1",           //3 #alias:freq_data_1
7, FIELD_TYPE_16B,   "freq2",           //3 #alias:freq_data_2
8, FIELD_TYPE_16B,   "counter1",        //3 #alias:c_data_1
9, FIELD_TYPE_16B,   "counter2",        //3 #alias:c_data_2
10, FIELD_TYPE_16B,  "counter3",        //3 #alias:c_data_3
11, FIELD_TYPE_8B,   "stop_state", // * //2
12, FIELD_TYPE_8B,   "d_state",         //2 #alias:sensordata
13, FIELD_TYPE_8B,   "snr_min",         //2
14, FIELD_TYPE_8B,   "snr_max",         //2
15, FIELD_TYPE_16B,  "ts_datai",        //2
16, FIELD_TYPE_16B,  "ts_data_0",       //2 #alias:ts_data
17, FIELD_TYPE_16B,  "ts_data_1",       //2 #alias:ow2_temp
18, FIELD_TYPE_16B,  "ts_data_2",       //2 #alias:ow3_temp
19, FIELD_TYPE_16B,  "ts_data_3",       //2 #alias:ow4_temp
20, FIELD_TYPE_32B,  "ibut_time",       //4 #alias:i_button_time
21, FIELD_TYPE_8A,   "ibut_id",         //8 #alias:i_button_id
22, FIELD_TYPE_16B,  "vbat_prc",        //8
23, FIELD_TYPE_16B,  "v_1224",          //8 #alias:innervoltage
24, FIELD_TYPE_32B,  "milage",          //8 #alias:milage


94, FIELD_TYPE_8B,  "fueltemp0",       //3
95, FIELD_TYPE_8B,  "fueltemp1",       //3
96, FIELD_TYPE_8B,  "fueltemp2",       //3
97, FIELD_TYPE_8B,  "fueltemp3",       //3
98, FIELD_TYPE_8B,  "fueltemp4",       //3
99, FIELD_TYPE_16B,  "fueldata0",       //3
100, FIELD_TYPE_16B, "fueldata1",       //3  #alias:fs_data_1
101, FIELD_TYPE_16B, "fueldata2",       //3  #alias:fs_data_2
102, FIELD_TYPE_16B, "fueldata3",       //3  #alias:fs_data_3
103, FIELD_TYPE_16B, "fueldata4",        //3 #alias:fs_data_4
```

```
109, FIELD_TYPE_16B, "acc_data_x",       //3 #alias:acc_x
110, FIELD_TYPE_16B, "acc_data_y",       //3 #alias:acc_y
111, FIELD_TYPE_16B, "acc_data_z",       //3 #alias:acc_z


140, FIELD_TYPE_32B, "can_log_s",
141, FIELD_TYPE_32B, "can_log_a",
142, FIELD_TYPE_32B, "can_log_b",
143, FIELD_TYPE_32B, "can_log_c",
144, FIELD_TYPE_32B, "can_log_d",
145, FIELD_TYPE_32B, "can_log_e",
146, FIELD_TYPE_32B, "can_log_f",
147, FIELD_TYPE_32B, "can_log_g",
148, FIELD_TYPE_32B, "can_log_r",
149, FIELD_TYPE_32B, "can_log_h",
150, FIELD_TYPE_32B, "can_log_i",
151, FIELD_TYPE_32B, "can_log_j",
152, FIELD_TYPE_32B, "can_log_k",
153, FIELD_TYPE_32B, "can_log_n",
154, FIELD_TYPE_32B, "can_log_o",
155, FIELD_TYPE_32B, "can_log_p",
156, FIELD_TYPE_32B, "can_log_u",
157, FIELD_TYPE_32B, "can_log_v",
158, FIELD_TYPE_32B, "can_log_wa",
159, FIELD_TYPE_32B, "can_log_wb",
160, FIELD_TYPE_32B, "can_log_wc",
161, FIELD_TYPE_32B, "can_log_wd",
162, FIELD_TYPE_32B, "can_log_we",
163, FIELD_TYPE_32B, "can_log_wf",
164, FIELD_TYPE_32B, "can_log_wg",
165, FIELD_TYPE_32B, "can_log_wh",
```

```
166, FIELD_TYPE_32B, "can_log_ta",
167, FIELD_TYPE_32B, "can_log_tb",
168, FIELD_TYPE_32B, "can_log_tc",
169, FIELD_TYPE_32B, "can_log_td",
170, FIELD_TYPE_32B, "can_log_te",
171, FIELD_TYPE_32B, "can_log_xa",
172, FIELD_TYPE_32B, "can_log_xb",
173, FIELD_TYPE_32B, "can_log_l",
174, FIELD_TYPE_32B, "can_log_m",
175, FIELD_TYPE_32B, "fm_fuel_lfc", // // #alias:fuel_lfc
176, FIELD_TYPE_32B, "fm_fuel_prc", // // #alias:fuel_prc
177, FIELD_TYPE_32B, "fms_rpm", // // #alias:rpm
178, FIELD_TYPE_32B, "fms_hours", // // #alias:hours
179, FIELD_TYPE_32B, "fms_milage", // // #alias:milage
180, FIELD_TYPE_32B, "fms_eng_tmp", // // #alias:engine_tmp
181, FIELD_TYPE_32B, "fms_amb_tmp", // // #alias:amb_tmp
182, FIELD_TYPE_32B, "fms_fuel_rt", // // #alias:fuel_rate
183, FIELD_TYPE_32B, "fms_fl_hrlf", // // #alias:fuel_hrlfc
184, FIELD_TYPE_32B, "fms_speed", // // #alias:speed
185, FIELD_TYPE_32B, "fms_eng_loa", // // #alias:engine_load
186, FIELD_TYPE_32B, "fms_axl_wgt", // // #alias:axle_wght
187, FIELD_TYPE_32B, "fms_srv_dst", // // #alias:srv_dst
188, FIELD_TYPE_32B, "fms_gros_wg", // // #alias:gross_wght
189, FIELD_TYPE_32B, "fms_taho_st", // // #alias:taho_st
190, FIELD_TYPE_32B, "fms_tah_spd", // // #alias:taho_speed
191, FIELD_TYPE_32B, "iqf_mt", // // #alias:iqf_mt
192, FIELD_TYPE_32B, "iqf_sp", // // #alias:iqf_sp
193, FIELD_TYPE_32B, "iqf_ambt", // // #alias:iqf_ambt
194, FIELD_TYPE_32B, "iqf_afzt", // // #alias:iqf_afzt
195, FIELD_TYPE_32B, "iqf_rpm", // // #alias:iqf_rpm
196, FIELD_TYPE_32B, "iqf_conf", // // #alias:iqf_conf
```

```
197, FIELD_TYPE_32B, "iqf_state", // // #alias:iqf_state

198, FIELD_TYPE_32B, "iqf_dr", // // #alias:iqf_dr

199, FIELD_TYPE_32B, "iqf_batv", // // #alias:iqf_batv

200, FIELD_TYPE_20Z, "imei",

201, FIELD_TYPE_20Z, "iccid1",

202, FIELD_TYPE_20Z, "iccid2",

203, FIELD_TYPE_8B,  "sim_num",    // #alias:sim_number

204, FIELD_TYPE_8B,  "srv_num",    // #alias:active_server

205, FIELD_TYPE_16Z, "op_name",

206, FIELD_TYPE_16B, "lac",

207, FIELD_TYPE_32B, "cell_id"

208, FIELD_TYPE_8B,  "gsm_power",  // #alias:gsmstrength


245, FIELD_TYPE_32B, "iqf_bata", // // #alias:iqf_bata

246, FIELD_TYPE_32B, "iqf_alc", // // #alias:iqf_alc

247, FIELD_TYPE_32B, "iqf_al1", // // #alias:iqf_al1

248, FIELD_TYPE_32B, "iqf_al2", // // #alias:iqf_al2

249, FIELD_TYPE_32B, "iqf_al3", // // #alias:iqf_al3

250, FIELD_TYPE_32B, "iqf_in"  // // #alias:iqf_in
```

Пример данных полей (без заголовка):

0B 00 03 EB 05 01 5A 09 02 01 00 5B 00 04 60 00 05 60 00 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF

0B = 11 - тип поля 11 - stop_state длина - 1 байт значение 00
03 = 3 тип поля 3 - vbat - длина - 2 байта значение 0x05EB = 1515
01 = 1 тип поля 1 - v_in длина - 2 байта значение 0x095A= 2394
02 = 2 тип поля 2 - ign_state длина 1 байт значение 1
00 = 0 тип поля 0 - alt длина 2 байта значение 005B = 91
04 = 4 тип поля 4 = adc1 длина 2 байта значение 0060 = 96
05 = 5 тип поля 5 = adc2 длина 2 байта значение 0060 = 96

FF = данных дальше нет. Разбор пакета можно остановить. Так же нет смысла разбирать пакет дальше если встретился неизвестный тип поля.

Порядок полей в пакете может быть произвольным, при этом общая длина пакета будет соответствовать настроенному набору полей.

Так как список полей в пакете 0x004C будет расширяться, то чтобы разбор пакетов в системе мониторинга не прерывался, необходимо при разборе пакетов выполнение 3 постулатов:

1. Если структура t_binary_container корректна (совпадают преамбула и контрольная сумма всего пакета), то пакет подтверждается - трекер считает, что все в порядке и готовит к отправке следующий пакет.
2. Если структура t_common_data_header внутри t_binary_container корректна (совпадают преамбулы и контрольные суммы), но packet_type системе мониторинга неизвестен, то переходим к разбору следующего пакета в контейнере (поле packet_len позволяет это сделать). Если структура t_common_data_header некорректна, то разбор контейнера на этом заканчивается.
3. Если система мониторинга встретила неизвестное поле внутри пакета B_PACKET_TYPE_L4_UNIVERSAL (0x004C), то разбор пакета на этом заканчивается, но все поля, которые удалось разобрать, добавляются в базу. Нужно учитывать, что пакет данного типа может быть вообще пустой. В этом случае в базу добавляются только поля из t_l4_data_base.

Тип пакета 0x81 - DDD файл с тахографа.

```
typedef struct {
  uint16_t file_id;    // id файла карты.
  uint8_t  file_type;  // тип файла карты
  uint16_t file_len;   // общая длинна файла
  uint8_t  count_mes;  // кол-во посылок с этим файлом
  uint8_t  number_mes; // номер посылки
} t_taho_data_header;

```

Компрессия данных

Новое устройство SAT-LITE 3 поддерживает компрессию отправляемых данных.

Пакет t_binary_container сжимается полностью по алгоритму LZ. К получившемуся бинарному массиву сверху цепляется вот такая структура (все упаковано):

```
typedef struct {
  uint16_t crc;
  uint16_t preamble;
  uint16_t data_len;
} t_compressed_header;
```

Преамбула другая:

#define B_COMPRESSED_PREAMBLE 0x9b3d

crc - контрольная сумма сжатого пакета
data_len - длина в сжатом виде

Т.е. на сервер передается сначала:
t_compressed_header
затем сжатый t_binary_container

Соответственно, алгоритм такой:

1. Проверяем что преамбула соответствует зашифрованному пакету
2. Проверяем контрольную сумму пакета
3. Расшифровываем (распаковываем) данные
4. Получившийся пакет обрабатываем как обычно

Пример сжатого пакета:

[26.01.17 18:28:24] Raw Data (binary): 9A 60 **3D 9B** **87 00** 03 EF FA 2C 8A 70 11 01 00 3F 01 67 D1 3D 53 61 31 8A 58 4C 00 08 00 00 B7 5F AA 40 BD 08 2B 01 00 02 0A 00 00 00 00 00 00 E8 3E 53 66 03 12 1D 09 03 05 1C C1 B3 3F 53 6B 03 18 1D DF 69 40 53 70 03 18 1D 1C F1 41 53 75 03 18 74 99 C5 42 53 7A 03 18 3A BA 93 43 53 7F 03 18 1D 2A 69 44 53 84 03 18 57 A0 80 46 53 89 03 12 1D 0C 03 05 1D 3A 3C 4F 53 B1 03 0F 1D 00 00 02 0B 03 05 1D 54 73 50 53 B6 03 18 1D

После распаковки получаем:

[26.01.17 18:28:24] Uncompressed Data (binary): EF FA 2C 8A 70 11 01 00 3F 01 67 D1 3D 53 61 31 8A 58 4C 00 08 00 00 B7 5F AA 40 BD 08 2B 01 00 02 0A 00 00 00 00 00 00 E8 3E 53 66 31 8A 58 4C 00 08 00 00 B7 5F AA 40 BD 08 2B 01 00 02 09 00 00 00 00 00 C1 B3 3F 53 6B 31 8A 58 4C 00 08 00 00 B7 5F AA 40 BD 08 2B 01 00 02 09 00 00 00 00 00 DF 69 40 53 70 31 8A 58 4C 00 08 00 00 B7 5F AA 40 BD 08 2B 01 00 02 09 00 00 00 00 00 1C F1 41 53 75 31 8A 58 4C 00 08 00 00 B7 5F AA 40 BD 08 2B 01 00 02 0A 00 00 00 00 00 99 C5 42 53 7A 31 8A 58 4C 00 08 00 00 B7 5F AA 40 BD 08 2B 01 00 02 09 00 00 00 00 00 BA 93 43 53 7F 31 8A 58 4C 00 08 00 00 B7 5F AA 40 BD 08 2B 01 00 02 09 00 00 00 00 00 2A 69 44 53 84 31 8A 58 4C 00 08 00 00 B7 5F AA 40 BD 08 2B 01 00 02 0A 00 00 00 00 00 A0 80 46 53 89 31 8A 58 4C 00 08 00 00 B7 5F AA 40 BD 08 2B 01 00 02 0C 00 00 00 00 00 3A 3C 4F 53 B1 31 8A 58 4C 00 08 00 00 B7 5F AA 40 BD 08 2B 00 00 02 0B 00 00 00 00 00 54 73 50 53 B6 31 8A 58 4C 00 08 00 00 B7 5F AA 40 BD 08 2B 00 00 02 0B 00 00 00 00 00

Далее обрабатываем как обычный пакет Лайт-3.

Библиотека сжатия описана в файлах LZ.c, LZ.h.