Reverse Engineering

Notes from various reverse engineering projects

Pax

Documentation on the hardware, firmware, and Bluetooth protocol used by the Pax Era, Era Pro, and Pax 3 vaporizers by Pax Labs.

Protocol Overview

This is mostly based off of reading the slightly deobfuscated source code of the Pax WebApp and Bluetooth packet logs.

All messages for controlling the device seem to go through a single service with UUID `8E320200-64D2-11E6-BDF4-0800200C9A66`. Of interest is a read characteristic with UUID `8E320201-64D2-11E6-BDF4-0800200C9A66` and write characteristic `8E320202-64D2-11E6-BDF4-0800200C9A66`; there is also a notification characteristic `8E320203-64D2-11E6-BDF4-0800200C9A66`. All commands are at least 16 bytes, but some commands can be longer. No padding is applied otherwise, as AES is used as if it were a stream cipher here. The first byte of each message is some sort of type indicator, which defines the format of the rest of the message's contents.

Blob data sent to/from the device appears to be AES-128 encrypted, in OFB mode. Every packet that is sent/received resets the cipher state, however, so there's no actual cipher state needed to be maintained between messages. The key for the cipher is derived during initialization from the device serial. All messages have a 16 byte IV at the end that is used for the OFB cipher to decode that message.

See this Python code for an example of how the encryption/decryption of packets is implemented.

There's also a more complete example on actually connecting to, and interfacing with a real device.

Note: This encryption mechanism only applies to the Pax Era and the Pax 3; the new Pax Era Pro has a more sophisticated protocol using nonces and AES-CTR mode, as well as a connection handshake that is not covered here.

Notifications

The notification characteristic is used relatively widely to indicate that new data is available. It appears to always send a single byte, which is the first byte of the encrypted packet that is ready to be sent.

Message Structure

Messages are read in blocks of at least 32 bytes (?). This is treated as two separate 16-byte values: the first is a standard device to host attribute message, while the latter is the 16-byte IV to use for cipher operations with this message. In all cases, the device key is used.

Device Key Derivation

The key used for encryption is derived by concatenating the serial number string (this is always 8 characters long) to itself to form a 16 byte blob. It is then encrypted using ECB with a fixed key (`F7 C8 66 C3 8F 78 75 30 86 29 3B D5 7D D3 25 40`), and the first 16 bytes are stored as the key with which all further messages are encrypted.

Message Types

Regardless of contents of the rest of the message, the first byte of the message is always reserved to indicate its type to allow correct decoding. Messages are simply packed structs; no special encoding is required. All multibyte integers are stored in little endian byte order. See here for a list of all message types.

Known message types are as follows:

LED brightness

Message type: `ATTRIBUTE_BRIGHTNESS`

```
struct {
    // 0 = 0%, 0x7F = 100%
    uint8_t brightness;
    // reserved (0) for Era
    uint8_t command;
};
```

Lock state

Message type: `ATTRIBUTE_LOCKED`

```
struct {
    // 0 = unlocked, 1 = locked
    uint8_t isLocked;
```

```
};
```

Device name

Message type: `ATTRIBUTE_DEVICE_NAME`

```
struct {
    // length of the string, in bytes
    uint8_t length;
    // name string. this is NOT null terminated
    char name[];
};
```

Heater set point

Message type: `ATTRIBUTE_HEATER_SET_POINT`

```
struct {
    uint16_t temp;
};
```

The temperature value is multiplied by 10; so 420°C is encoded as 4200 or 0x1068.

Status update

Message type: `ATTRIBUTE_STATUS_UPDATE`

This message, when sent to the device, triggers the transmission of all attribute values whose bits are set. For example, to read both `ATTRIBUTE_ACTUAL_TEMP` (1) and `ATTRIBUTE_CHARGE_STATUS` (7) simultaneously, the types value would be set to `0b10000001`.

```
struct {
    // bitmask of all attribute types to read out
    uint64_t types;
};
```

Message Types

This is a list of all message types as extracted from the Pax mobile application. The names are identical to what the app calls them.

Name	Description	ID	Era		Pax 3	
			Read	*Write*	*Read*	*Write*
ATTRIBUTE_ACTUAL_TEMP	Actual temperature of the heater	1	✓	✓	✓	✓
ATTRIBUTE_HEATER_SET_POINT	Set temperature of the device's heater	2	✓	✓	✓	✓
ATTRIBUTE_BATTERY		3	✓	✓	✓	✓
ATTRIBUTE_USAGE		4	✓	✓	✓	✓
ATTRIBUTE_USAGE_LIMIT		5	✓	✓	✓	✓
ATTRIBUTE_LOCKED	Whether the user interface of the device is locked or not	6	✓	✓	✓	✓
ATTRIBUTE_CHARGE_STATUS		7	✓	✓	✓	✓
ATTRIBUTE_POD_INSERTED		8	✓	✓	✓	✓
ATTRIBUTE_TIME		9	✓	✓	✓	✓
ATTRIBUTE_DEVICE_NAME	Display name of the device	10	✓	✓	✓	✓
ATTRIBUTE_REPLAY		13	✓	✓	✓	✓

ATTRIBUTE_GAME_MODE		15	□	□	□	□
ATTRIBUTE_HEATER_RANGES		17	□	□	□	□
ATTRIBUTE_LOG_SYNC_REQUEST		18	□	□	□	□
ATTRIBUTE_DYNAMIC_MODE	Current dynamic heater profile	19	□	□	□	□
ATTRIBUTE_COLOR_THEME	Selects a fixed color scheme, or sets the raw RGB values, depending on device.	20	□	□	□	□
ATTRIBUTE_BRIGHTNESS	Brightness of user interface LEDs	21	□	□	□	□
ATTRIBUTE_HAPTIC_MODE		23	□	□	□	□
ATTRIBUTE_SUPPORTED_ATTRIBUTES	Indicates which properties are supported by the device; this is a 64-bit wide bitfield.	24	□	□	□	□
ATTRIBUTE_HEATING_PARAMETERS		25	□	□	□	□
ATTRIBUTE_UI_MODE		27	□	□	□	□
ATTRIBUTE_SHELL_COLOR	Color of the outside of the device (?)	28	□	□	□	□
ATTRIBUTE_LOW_SOC_MODE		30	□	□	□	□
ATTRIBUTE_CURRENT_TARGET_TEMP	Target temperature of the internal heater control loop	31	□	□	□	□

ATTRIBUTE_HEATING_STATE	State of the heater	32	☐	☐	☐	☐
ATTRIBUTE_SESSION_CONTROL		36	☐	☐	☐	☐
ATTRIBUTE_HAPTICS	Control over the vibration motor	40	☐	☐	☐	☐
ATTRIBUTE_LOG_REQUEST		41	☐	☐	☐	☐
ATTRIBUTE_POD_DATA		42	☐	☐	☐	☐
ATTRIBUTE_ENCRYPTION_EXCHANGE	Seems to be used by Era Pro to negotiate a session key	49	☐	☐	☐	☐
ATTRIBUTE_ENCRYPTION_PACKET	Seems to be used by Era Pro to negotiate a session key	50	☐	☐	☐	☐
ATTRIBUTE_BLE_DIS_DATA		52	☐	☐	☐	☐
ATTRIBUTE_FIND_MY_PAX		54	☐	☐	☐	☐
ATTRIBUTE_STATUS_UPDATE	Request current values for the given attributes; the payload is a 64-bit bitfield, encoded identically to the supported attributes query.	254	☐	☐	☐	☐

At least the names and IDs are known to be accurate. The descriptions were determined through reverse engineering the code, and the device support was also determined from seeing which packets the device sent at what times.