BasicMAC FUOTA

BasicMAC offers a defragmentation service (`"fuota"`) to reconstruct firmware updates from redundantly received fragments. Once fully reconstructed, the application can verify the digital signature of an update and pass it on to the *BasicLoader* boot loader. *BasicLoader* can process and install update files (`.up`) generated by `zfwtool`.

Firmware Update Files

Firmware update files (`.up`) are created from one or more firmware files (`.zfw`) using `zfwtool`. A firmware update file contains a compressed firmware image and can be either self-contained or a delta to a previous firmware. Optionally the update can be appended with a digital signature. The firmware files (`.zfw`) are generated by the `make` process in the build directory and contain a binary firmware image plus some meta information.

- **generate self-contained firmware update file**

  ```
  python zfwtool.py mkupdate -s mykey.pem --passphrase mysecret firmware.zfw firmware.up
  ```

- **generate delta firmware update file**

  ```
  python zfwtool.py mkupdate -s mykey.pem --passphrase mysecret firmware2.zfw -d firmware1.zfw delta-1-2.up
  ```

Code-Signing Keys

To sign the firmware update with `zfwtool` a code-signing key pair is required. The private key must be kept secret to the application owner and resides in a passphrase-protected `.pem` file. The public key must be embedded in the application and is used as trust anchor to verify the signature of a received update file. The key pair can be created and managed using *OpenSSL*.

- **generate EC key pair in passphrase-protected `.pem` file**

  ```
  openssl ecparam -name prime256v1 -genkey | openssl ec -out mykey.pem -aes256 -passout pass:mysecret
  ```

- **print public key from passphrase-protected `.pem` file (last 64 bytes of DER struct as C byte array)**

  ```
  openssl ec -in mykey.pem -passin pass:mysecret -pubout -outform der | xxd -i -s 27 -c 16
  ```

- embed output in C code (`fuota.c`)

```
static const unsigned char pubkey[64] = {
    0xXX, 0xXX, 0xXX, 0xXX, 0xXX, 0xXX, 0xXX, 0xXX, 0xXX, 0xXX, 0xXX, 0xXX, 0xXX, 0xXX, 0xXX, 0xXX,
    0xXX, 0xXX, 0xXX, 0xXX, 0xXX, 0xXX, 0xXX, 0xXX, 0xXX, 0xXX, 0xXX, 0xXX, 0xXX, 0xXX, 0xXX, 0xXX,
    0xXX, 0xXX, 0xXX, 0xXX, 0xXX, 0xXX, 0xXX, 0xXX, 0xXX, 0xXX, 0xXX, 0xXX, 0xXX, 0xXX, 0xXX, 0xXX,
    0xXX, 0xXX, 0xXX, 0xXX, 0xXX, 0xXX, 0xXX, 0xXX, 0xXX, 0xXX, 0xXX, 0xXX, 0xXX, 0xXX, 0xXX, 0xXX,
};
```

FUOTA Session

A FUOTA session is the process of reconstructing the original firmware update file on the device from subsequently received redundant fragments. The session state consists of the stored fragment data, a redundancy matrix, and the session parameters *session-id*, *frag-size* and *frag-cnt*. During a FUOTA session the session parameters must not be changed! The session state is stored persistently, so the FUOTA session can be resumed after the device has been reset.

session-id

Content identifier for the session in progress (e.g. firmware version).

frag-size

Size of the fragments (multiple of 4).

frag-cnt

Total number of fragments in the update file. *frag-cnt* = (*updatelen* + *frag-size* - 1) / *frag-size*.

Fragmentation

To deliver a firmware update over LoRaWAN, the update file (`.up`) needs to be split into smaller fragments and must be redundancy-encoded. The size of the fragments (*frag-size*) must be chosen so that it always fits in the allowed LoRaWAN payload size of the region and data rate in use. Additionally the fragment size mut be a multiple of 4 bytes. See example script `fragger.py` for generation of fragment data and downlink payloads.

Example Application ex-fuota

The example application sends periodic uplinks with application data on port 15 every 60 seconds. When it receives a downlink on port 16, the payload is interpreted as FUOTA message and is processed accordingly. On reception of a FUOTA message three uplinks with the current FUOTA status will be generated on port 16 in an interval of 10 seconds. This way the application server can monitor progress and gets additional downlink opportunities to deliver more fragments.

When enough fragments have been received to fully reconstruct the original update file, the digital signature is validated and the update is passed on to *BasicLoader* and a device reset is triggered. After the reset the boot loader will install the update and start the new firmware.

In addition to the fragment data the FUOTA message in the downlink also holds a small header with the FUOTA session parameters and an index describing the redundancy matrix of the fragment data.

The session parameters *frag-size*, *frag-cnt*, *src-crc* and *dst-crc* must not change during a FUOTA session! When these parameters change, the current session is discarded and a new session with the new parameters is started.

NOTE: All multi-byte integers (including CRCs) are encoded LSBF!

Downlink Format (port 16)

Field	Offset (bytes)	Size (bytes)	Description
src-crc	0	2	short CRC of delta firmware or 0
dst-crc	2	2	short CRC of updated firmware
frag-cnt	4	2	total number of fragments in update file
frag-idx	6	2	fragment index (redundancy matrix generator)
frag-data	8	*frag-size*	fragment data...

Uplink Format (port 16)

Field	Offset (bytes)	Size (bytes)	Description
fw-crc	0	4	CRC of current firmware
done-cnt	4	2	number of reconstructed fragments
frag-cnt	6	2	total number of fragments in update file