OSNMAlib Improvements and Real-Time Monitoring of Galileo OSNMA

Aleix Galan
ESAT
KU Leuven
Leuven, Belgium
0000-0002-5762-6982

Cristian Iñiguez
Dept. of Telecommunications
UAB
Barcelona, Spain
0009-0000-8950-2776

Ignacio Fernandez-Hernandez
DG DEFIS
European Commission
Brussels, Belgium
0000-0002-9308-1668

Sofie Pollin
ESAT
KU Leuven
Leuven, Belgium
0000-0002-1470-2076

Gonzalo Seco-Granados
Dept. of Telecommunications
UAB
Barcelona, Spain
0000-0003-2494-6872

2024 International Conference on Localization and GNSS (ICL-GNSS) | 979-8-3503-8078-1/24/$31.00 ©2024 IEEE | DOI: 10.1109/ICL-GNSS60721.2024.10578487

***Abstract*—Galileo will declare OSNMA (Open Service Navigation Message Authentication), a civil GNSS signal authentication scheme, operational in the near future. OSNMAlib, an open-source library that implements OSNMA, was presented two years ago after the test phase of the protocol started and has since undergone several upgrades. In this paper, we disclose these upgrades, which comprise new input sources, new features and optimizations, and the creation of an OSNMA real-time monitoring website. For each input source (SBF, UBX, GNSS-SDR and galmon), we describe how can they be integrated with an OSNMA library and what pitfalls to avoid. The new features include data retrieval and time to first authenticated fix optimizations and a new logging format aimed at researchers. This logging format is used in *osnmalib.eu* website to display, in a friendly and understandable way, the live Galileo and OSNMA messages and the OSNMAlib authentication output. Additionally, the website also provides the I/NAV data bits to help snapshot receivers and other GNSS-based applications.**

***Index Terms*—Global navigation satellite system, Galileo, OSNMA, OSNMAlib, Authentication**

I. Introduction

Galileo is the first GNSS constellation to add authentication to their civil signals. The authentication is meant to protect the users against spoofing, the transmission of forged GNSS-like signals that induce a wrong position and time fix. This protection is offered at navigation message level, using the Galileo OSNMA (Open Service Navigation Message Authentication) protocol.

OSNMA is a cryptographic protocol that leverages the unpredictability of the transmitted bits and the verification of these bits at a later time, a variation of the TESLA (Timed Efficient Stream Loss-Tolerant Authentication) protocol [1]. A spoofer crafting a false signal is not able to guess the cryptographic material, but any receiver can verify the correctness of the bits once received. These types of Navigation Message Authentication (NMA) techniques have already been described over the last decades [2]. However, their adoption was held back due to the need to modify the navigation message structure to accommodate them [3] [4]. For the last two years, OSNMA has been transmitted in the Galileo E1-B signal in test mode, and it is expected to be declared operational in the near future [5].

At the start of the test transmission, we presented OSNMAlib [6] [7], an open-source OSNMA library. The library, written in Python, can easily be used for research purposes and to add OSNMA support to GNSS receivers. Now, with the imminent operational declaration of the service, we present the library improvements. The enhancements are focused on optimizing data extraction in harsh environments, improving the time to first authenticating fix, the addition of new GNSS input sources, and a refreshed logging system.

Over the last years, other open-source OSNMA implementations have emerged. A well-maintained and interesting implementation is *galileo-osnma* [8] written in Rust and aimed for embedded devices, released under a permissive license. Another notable implementation that is also open-source is *fgi-osnma* [9], by the Finnish Geospatial Research Institute, which allows to integrate OSNMA to their GNSS software-defined receiver.

In this paper, we present the newly added input sources to OSNMAlib (SBF, UBX, GNSS-SDR and galmon) and give recommendations to other libraries willing to incorporate the same input sources. Then, we disseminate the new optimizations and logging features implemented in OSNMAlib. Finally, we introduce a real-time OSNMA monitoring website [10] that we build using OSNMAlib, which includes the exposure of navigation message bits.

This research was partially funded by the Research Foundation Flanders (FWO) Frank de Winne PhD Fellowship, project number 1SH9424N (Aleix Galan).

II. OSNMA AND OSNMALIB

Galileo OSNMA is transmitted within the Galileo I/NAV message structure for the E1-B signal. The I/NAV message is structured in 15 two-second nominal pages of 240 bits each that are grouped in a 30-seconds sub-frame. The nominal pages are divided into even and odd pages of 120 bits each, and together transmit a Word Type (WT). The WTs encapsulate Galileo navigation data and are what OSNMA authenticates. The page structure is represented in Fig. 1.

The OSNMA data message is transmitted in batches of 40 bits on each nominal page. These bits are concatenated at the end of each sub-frame to create complete OSNMA structures. These structures are the HKROOT (Header and Root Key), with 120 bits, which transmits the long-term authentication data to verify the keys; and the MACK (MAC and Key), with 480 bits, which transmits the authentication tags for the navigation data and the authentication key. For further details, the complete OSNMA specification is available in the OSNMA SIS ICD [11] and in the receiver guidelines [12].

An external library willing to process OSNMA needs first access to the navigation data message, or at least to the bits containing the WTs and the 40 bits of OSNMA message, and the Satellite Vehicle ID (SVID) to which the navigation data belongs. Additionally, OSNMA needs to know the Galileo Satellite Time (GST) computed by the GNSS receiver to validate some protocol time constraints. Moreover, in an offline scenario, accessing the receiver computed time is the only way to post-process OSNMA.

In OSNMAlib, the input data is encapsulated in an object that contains the I/NAV message data from E1-B (240 bits), the GST in Time Of Week (TOW) and Week Number (WN) at the beginning of the page transmission, and the SVID. Any OSNMAlib input source must be created as a Python iterator abstraction that returns one of these objects at every iteration of the library reception loop. Using this technique, the origin

Even/odd=1	Page Type	Data j (2/2)	OSNMA	SAR	Spare	CRC_j	SSP	Tail	Total (bits)
1	1	16	40	22	2	24	8	6	120

Even/odd=0	Page Type	Data k (1/2)	Tail	Total (bits)
1	1	112	6	120

Fig. 1. Nominal page structure from the Galileo I/NAV message transmitted in the E1-B signals, from the Galileo ICD [13].

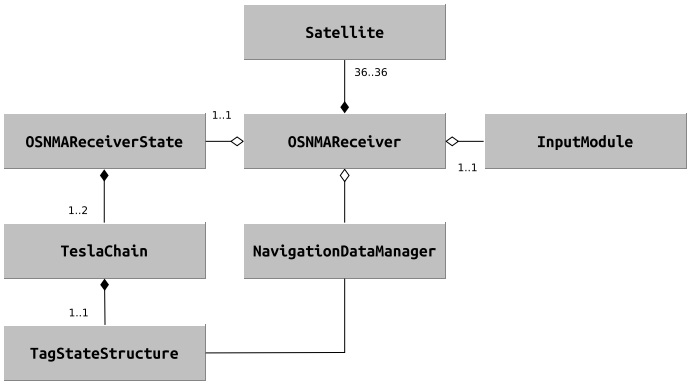

Fig. 2. Simplified UML diagram of OSNMAlib. The `InputModule` is an abstract class that needs to be particularized for each input type. There may be 2 `TeslaChain` objects when the TESLA Key is being renewed.

of the input data does not matter for OSNMAlib and can come from heterogeneous sources.

A simplified architecture of OSNMAlib is shown in Fig. 2. As previously discussed, the library gets the input data from an iterator: this is done in the `OSNMAReceiver` module. This module also sends the navigation pages to the `NavigationDataManager` and reconstructs the OSNMA messages distributed in multiple pages and sub-frames. For this last purpose, it uses the Satellite class. Then, the OSNMA messages are relied to the `OSNMAReceiverState` class that interprets them and keeps the internal state of the library. Based on the OSNMA messages information, the `OSNMAReceiverState` class decides to call the different OSNMA functions such as update the public key, create a new TESLA chain, authenticate a TESLA key, verify authentication tags, etc. The authentication tags are stored in the class `TagStateStructure` that belongs to a `TeslaChain`, because the configuration of the TESLA chain in force defines the format of the tags. Finally, when an authentication tag needs to be verified, the `TagStateStructure` class requests the necessary navigation data to the `NavigationDataManager`.

OSNMAlib allows the user to configure several parameters of the execution; one of the most important is the Time Lag (TL) value, which defines the synchronization lag between the receiver sourcing the navigation data and the GST. The security of the OSNMA protocol can only be guaranteed if this value is known because it defines which authentication tags can be safely used for authenticating navigation data. Another relevant configuration parameter available in OSNMAlib is the re-use of already available cryptographic data to speed up the bootstrap of OSNMA (the so-called *warm start* and *hot start* procedures).

III. DATA INPUT SOURCES

The data input sources for OSNMAlib have been expanded substantially since the start of the Galileo OSNMA test phase. This section provides a detailed description on how to add these input sources to any OSNMA library and the problems

one may encounter. In addition to the inputs listed here, OSNMAlib supports the official OSNMA test vector format [12]. Moreover, in the OSNMAlib repository, we provide guidance on how to develop and integrate new input sources.

A. Septentrio: SBF

SBF is the binary output format of Septentrio receivers. This format encapsulates semantically similar data into SBF blocks. All blocks have a header with, among others, the block type and a time stamp in WN and TOW, which interpretation is block dependent.

The interesting SBF block for an OSNMA library is the `GALRawINAV` block. This block contains 234 bits of an I/NAV navigation page, which results from the concatenation of the even and odd sub-pages minus the six trail bits. The block also contains the SVID and signal from which the message was received. Finally, the time stamp in the block header corresponds to the time of transmission of the last bit in GPS time.

To adapt the SBF format to OSNMA input we first have to correct the time stamp. OSNMA expects the time in GST convention and referring to the transmission of the first bit of the navigation message. Therefore, we must subtract 1024 weeks from the WN to change from GPS time to GST and two seconds to the ToW to move to the beginning of the page. Next we have to regenerate the 240 bits that OSNMAlib expects by adding the six removed trail bits required for the convolutional code, which must be zero-filled. Finally, OSNMAlib currently only supports E1-B, so the blocks need to be filtered by the signal indicator.

We developed an OSNMAlib input module for three SBF sources: a file, a TCP client and a TCP server. The file source allows to post-process any log file that contains the `GALRawINAV` block. The TCP sources (client and server) are developed with real-time processing in mind, because Septentrio receivers can serve SBF from or to TCP ports.

B. u-blox: UBX

The u-blox receivers binary output format is called UBX and consists of frames that contain different data. Even though UBX has the `RXM-SFBRX` frame that provides the I/NAV page bits and SVID, this frame lacks the TOW and WN, which are essential for the correct functioning of OSNMAlib. To address this issue, we can use other frames from the UBX protocol that include both TOW and WN, such as the TIM-TP, NAV-TIMEGAL, NAV-TIMEUTC, NAV-TIMEGPS, and NAV-TIMEBDS. Note that the time information extracted from these frames needs to be converted to GST.

If OSNMAlib has no access to the time frames, for example if a file was recorded without them, both TOW and WN must be calculated. For a real-time case, the computer clock can be utilized to precisely calculate the TOW and WN. This strategy requires to first initialize both TOW and WN with a value transmitted in one of the word types, and subsequently update the time using the computer clock. On the other hand, in the case of UBX files, the absence of time frames complicates the exact calculation of the TOW and WN, and need to be approximated. The different possibilities when using a UBX output stream are shown in Fig. 3.

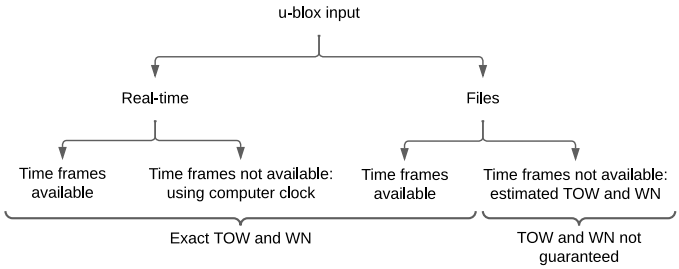

Fig. 3. OSNMAlib calculation of GST for the u-blox receiver input source depending on the available data.

The proposed strategy when post-processing a file without time frames is based on the known sequence of word types in a sub-frame, as defined in the Galileo ICD [13]. The algorithm starts by extracting the TOW from a WT 0, 5 or 6 and then updates the TOW based on the next WT received using a lookup table. The lookup table (shown in Table I) indicates the seconds elapsed between the previous and current WTs. Therefore, if pages are lost, the receiver can determine how much time has passed between them. Of course, when a WT containing a TOW is received, the internal TOW is verified and adjusted.

However, this process lacks accuracy and may fail to produce correct results. The first problem is that the WT sequence described in the Galileo ICD is indicative an may have deviations. A known source of deviations are satellites with the health status bits in the navigation message set different than 0. The WT from these satellites need to be excluded from the algorithm. The second problem is that, if the receiver does not receive pages for more than one sub-frame, the estimation of the first new pages received can be inaccurate. Therefore, whenever possible, it is advisable to use the aforementioned time frames.

TABLE I
LOOKUP TABLE WITH SECONDS ELAPSED BETWEEN A PRIOR NOMINAL PAGE AND A CURRENT ONE ACCORDING TO THE GALILEO ICD [13].

Prior WT	Current WT: 2	4	7 9	8 10	17 18	19 20	16	22	1	3
2	30	2	6	8	10	12	14	18	20	22
4	28	30	4	6	8	10	12	16	18	20
7/9	24	26	30	2	4	6	8	12	14	16
8/10	22	24	28	30	2	4	6	10	12	14
17/18	20	22	26	28	30	2	4	8	10	12
19/20	18	20	24	26	28	30	2	6	8	10
16	2	4	8	10	12	14	14	4	6	8
22	12	14	18	20	22	24	10	30	2	4
1	10	12	16	18	20	22	8	28	30	2
3	8	10	14	16	18	20	6	26	28	30
0	4	6	8	10	12	14	2	2	4	6
5	6	8	10	12	14	16	4	24	26	28
6	26	28	30	4	6	8	10	14	16	18

C. GNSS-SDR

GNSS-SDR [14] is an open-source GNSS software-defined receiver written in C++ that supports multiple constellations, including Galileo, and real-time kinematics (RTK) positioning. The receiver is highly configurable using a text-based file and completely independent of the radio-frequency front-end used. Therefore, any user can record a GNSS signal (e.g. with a software-defined radio), process it with the receiver (in real-time or in post-process), and obtain a precise position and time fix in a variety of output formats.

To integrate GNSS-SDR to OSNMAlib we need to access the navigation message bits, which in GNSS-SDR is done using what they call a telemetry decoder block. The software offers several telemetry decoders, but we are interested in the `Galileo_E1B_Telemetry_Decoder` that decodes the INAV message of E1-B signal component. To retrieve the decoded bits we use another of the receiver features, which sends the output information by the telemetry decoders to an IP and UDP port. The configuration needed in the GNSS-SDR configuration file to enable all these features is presented in Listing 1.

The navigation messages are sent to the UDP port encapsulated using the *protobuf* [15] standard. Each protobuf structure contains the GNSS constellation, signal and SVID to which the bits belong, the TOW of the last symbol received, and the navigation message bits. For Galileo I/NAV, each message contains the 120 bits of a half page (even or odd).

The first issue when integrating GNSS-SDR's output to OSNMAlib is the lack of WN in the protobuf structure. We solved this by discarding the navigation message received until we receive a Word Type 0 or 5 and extract the WN from it. We also provide the users with an option to set the WN themselves. The second issue is that OSNMAlib works with full pages, not with half pages. Therefore, we have to save the even page in memory until the corresponding odd page is received before concatenating and transferring them to the library. Finally, we need to offset the TOW to align it with the beginning of the full page.

```
1 TelemetryDecoder_1B.implementation =
      Galileo_E1B_Telemetry_Decoder
2 NavDataMonitor.enable_monitor = true
3 NavDataMonitor.client_addresses = 127.0.0.1
4 NavDataMonitor.port = 1234
```

Listing 1. GNSS-SDR configuration needed to extract Galileo E1-B navigation data bits and send them to a UDP port at 127.0.0.1:1234.

D. Galmon network

The *galmon network* [16] is an open-source project that aggregates GNSS data from receivers around the globe for monitoring purposes. They have a dashboard for the main GNSS constellations and signals, with live information about the satellite status and data transmitted. Additionally, they have recently introduced a historical data dashboard. Any user can contribute or receive data from the distributed network; hence it is an attractive way to use OSNMAlib without any receiver.

Explicitly for disseminating OSNMA data bits, they facilitate the endpoint `86.82.68.237:10000` where every I/NAV message being send in the network is redirected. The navigation messages are again encapsulated using the protobuf standard but in a different format than the one mentioned in III-C.

Each protobuf structure has a header with metadata about the galmon network's receiver and the type of message sent. We are interested in the Galileo I/NAV structure, which contains the GNSS constellation, signal and SVID from where the navigation message bits were retrieved, the time information (WN and TOW), and the proper I/NAV bits. However, the I/NAV bits are fragmented in multiple fields instead of being a continuous 240-bits stream.

To integrate galmon with OSNMAlib, we do not need to reconstruct the full I/NAV message; we only need the field named `contents`, which stores the navigation word data, and the field named `reserved1`, which stores the 40 OSNMA bits. The rest of the 240 bits can be set to 0.

There are two considerations consider when integrating galmon with an OSNMA library. The first is that, at the time of writing this paper, the algorithm used by the network to estimate the TOW of pages coming from u-blox receivers does not take into consideration the latest changes in the Galileo ICD. Currently, the word type 16 is transmitted in the eighth and fifteenth page positions, but previously it was only transmitted in the eighth. Thus, it always gets assigned a page eight TOW. The solution implemented in OSNMAlib (described in Fig. 4) is to keep in memory a TOW counter to check if the word type 16 is transmitted in the new position with the wrong TOW value.

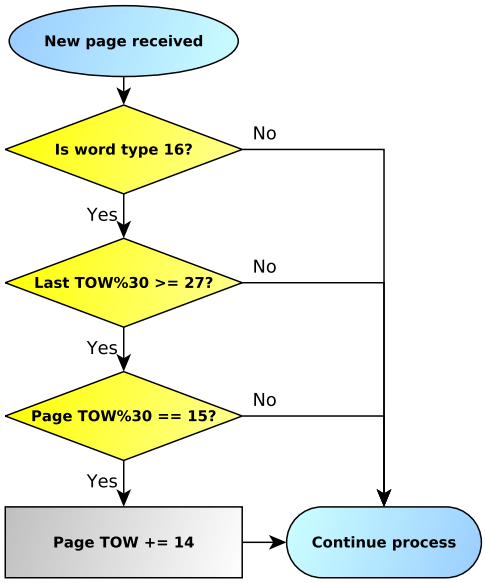

Fig. 4. Fix at library level to correct the assignation by galmon of the TOW for the WT 16.

The second consideration is that the messages in the network do not necessarily follow a chronological order, some receivers send messages delayed by a few seconds. Also, the same page can be received multiple times coming from different receivers in the network. OSNMAlib filters the messages to not use twice the same page, and to use only the most recent ones.

IV. New Features

A. Protocol optimizations

To optimize the retrieving of OSNMA data in environments with fading, such as urban scenarios, OSNMAlib implements a page-level processing of partially received sub-frames. This strategy was first described in [17] and greatly improves the Time To First Authenticated Fix (TTFAF) and continuous authentication of navigation data.

This approach allows the extraction of each authentication tag individually instead of parsing a fully received sub-frame. In this way, even if only two pages out of the fifteen that form the sub-frame are correctly received, it may be possible to obtain tags from them. The position of each tag in the tag sequence shall still be verified.

The page-level processing also enables to reconstruct a sub-frame's TESLA key from pages belonging to multiple satellites. All satellites transmit the same TESLA key at the same epoch; therefore, if OSNMAlib is not able to receive a complete TESLA key from any satellite but receives enough fragments in different pages, it can regenerate the key.

Another optimization in OSNMAlib is the link between authentication tag and navigation data to be authenticated using the new Cut-off Point (COP) field. OSNMAlib already implemented an intelligent way of linking tag and data using the Issue of Data (IOD) value to merge navigation data belonging to multiple sub-frames. Now, with the introduction of the COP field in the last OSNMA protocol version, this optimization is taken further allowing to safely assume that the data from one sub-frame was also transmitted in the previous. This assumption allows to get times to first authenticated fix in hot start as low as 44 seconds [18].

B. OSNMAlib sub-frame logging

The first version of OSNMAlib had only one logging stream that reported all OSNMA events in chronological order. That is, every time a new page was processed, everything triggered by the information on that page was reported. Although this logging stream provides interesting information for debugging purposes, it tends to be too verbose.

With the extraction of relevant metrics from OSNMA in mind, we developed a new logging stream that reports at the end of every sub-frame (i.e. every 30 seconds) on the state of OSNMAlib. This logging option presents in a clear and readable format the following information per sub-frame: navigation data received, OSNMA data received, OSNMA status, and authenticated data. Fig. 5 shows an example of how the information is displayed in a console. Additionally, the logging stream can be configured to log in pure *json* format for an easy ingest in any tool willing to process it. Both logging options can be enabled and disabled independently, and both can log to console or to a file.

```
INFO - --- STATUS END OF SUBFRAME GST 1267 35460 ---

OSNMAlib Status: STARTED

## Nav Data Received in the Subframe
{'04': {'ADKD0': {1: None, 2: None, 3: None, 4: None, 5: 'W5'}, 'ADKD4': {6: 'W6', 10: None}},
 '05': {'ADKD0': {1: 'W1', 2: 'W2', 3: 'W3', 4: 'W4', 5: None}, 'ADKD4': {6: 'W6', 10: None}},
 '09': {'ADKD0': {1: 'W1', 2: 'W2', 3: 'W3', 4: 'W4', 5: 'W5'}, 'ADKD4': {6: 'W6', 10: None}},
 '15': {'ADKD0': {1: None, 2: None, 3: 'W3', 4: 'W4', 5: 'W5'}, 'ADKD4': {6: 'W6', 10: None}},
 '31': {'ADKD0': {1: 'W1', 2: None, 3: 'W3', 4: 'W4', 5: 'W5'}, 'ADKD4': {6: None, 10: None}},
 '34': {'ADKD0': {1: 'W1', 2: 'W2', 3: None, 4: None, 5: None}, 'ADKD4': {6: 'W6', 10: None}}}

## OSNMA Data Received in the Subframe
{'04': {'Tags': [None, None, None, None, None, None], 'Key': None},
 '09': {'Tags': [[9, 0, 15], [31, 0, 15, 'FLX'], [9, 4, 15], [5, 0, 15, 'FLX'], [9, 12, 15], [21, 0, 15]],
        'Key': {'Value': '3f76d088c4e3e7ceaa294f959e9f5ca8', 'Verified': True, 'Reconstructed': False}},
 '15': {'Tags': [None, [5, 0, 15, 'FLX'], None, None, None, None],
        'Key': {'Value': '3f76d088c4e3e7ceaa294f959e9f5ca8', 'Verified': True, 'Reconstructed': True}},
 '34': {'Tags': [None, None, None, None, None, None], 'Key': None}}

## OSNMA Status
{'Tesla Chain in Force': {'NMAS': 'TEST',
                          'CID': 0,
                          'CPKS': 'NOMINAL',
                          'PKID': 1,
                          'HF': 'SHA_256',
                          'MF': 'HMAC_SHA_256',
                          'KS': 128,
                          'TS': 40,
                          'MACLT': 34,
                          'MACLT Sequence': [['00S', 'FLX', '04S', 'FLX', '12S', '00E'],
                                             ['00S', 'FLX', '00E', '12S', '00E', '12E']]},
 'Public Key in Force': {'NPKID': 1, 'NPKT': 'ECDSA_P256', 'MID': 0}}

## OSNMA Authenticated Data
{'ADKD0': {'09': {'iod': '0000111001', 'start_gst': [1267, 35430], 'last_gst': [1267, 35430], 'acc_length': 40},
           '05': {'iod': '0000111000', 'start_gst': [1267, 35430], 'last_gst': [1267, 35430], 'acc_length': 120},
           '34': {'iod': '0000111001', 'start_gst': [1267, 35430], 'last_gst': [1267, 35430], 'acc_length': 40}},
 'ADKD4': {},
 'ADKD12': {}}
```

Fig. 5. Example of the OSNMAlib status logging in a urban scenario with severe fading. During the sub-frame at GST 1267 35460 multiple pages were lost, which impacted in the number of words extracted and OSNMA data. It also displays the last authenticated OSNMA status information and the currently authenticated navigation data.

V. OSNMA Status Website

A. OSNMAlib dashboard

Using the new sub-frame status logging of OSNMAlib described in IV-B we have created a website *osnmalib.eu* [10] to monitor in real-time the status of OSNMA. Currently, the website runs two parallel OSNMAlib instances: one processing data from a Septentrio mosaic-X5 receiver [19] located at KU Leuven, Belgium; and the other processing global aggregated data from the galmon network.

After each sub-frame, the dashboard displays the last authenticated OSNMA status information, the number of satellites in view and the subset of those transmitting OSNMA, the information obtained from each satellite, and the data authentication output. The information retrieved for a satellite is presented as shown in Fig. 6. The navigation data word types are grouped per authentication tag type, and the OSNMA tags indicate for which satellite is the tag, the COP value, and if it is a flex tag. The tag color indicates the tag type: blue for ADKD0, orange for ADKD4, and pink for ADKD12. The TESLA key received is provided under the tags.

The accumulated navigation data information for each satellite is displayed as shown in Fig. 7. Each color represents a tag type in the same sequence mentioned above. The first column

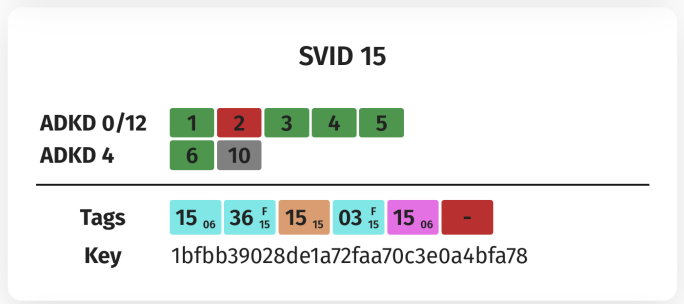

Fig. 6. Data blocks obtained from satellite 15 in the last sub-frame as displayed in the OSNMAlib website.

15	0000101111	1280 336210	200 bits
	N/A	1280 336180	7200 bits
	0000101110	1280 335910	600 bits

Fig. 7. Navigation data authentication status for satellite 15 as displayed in the OSNMAlib website.

Fig. 8. Example of the I/NAV bits as retrieved from the *osnmalib.eu* website. The *json* structure contains the last 12 sub-frames received by OSNMAlib, and each sub-frame contains the WN, the TOW and the bits from each satellite organized in 240 bit pages.

contains the Issue of Data (IOD) to identify the navigation message being authenticated, except for ADKD4 whose words do not contain an IOD. The next column indicates the GST of the last verified tag for the navigation data. The last column contains the accumulated bit-length of tags verified for the navigation data, an indirect metric to the number of tags verified.

B. Navigation message bits endpoint

Another feature of the *osnmalib.eu* website is the exposure of the raw I/NAV message as received by OSNMAlib in *json* format through the `/septentrio/subframe_bits` and `/galmon/subframe_bits` endpoints. The navigation message bits are provided per sub-frame, identified by the WN and TOW, and organized per satellite. Each satellite contains 15 pages of 240 bits with the navigation message in hexadecimal. If a page is lost the value for the page is `null`. Fig. 8 shows how the *json* structure looks like.

The endpoints return the last 12 sub-frames bits from the selected receiver. The decision of saving 12 sub-frames is to always have in memory an ADKD12 tag and the applicable TESLA key (transmitted with an additional delay of 10 sub-frames).

Snapshot GNSS receivers may use these navigation data bits to compare the unpredictable OSNMA bits they receive with the OSNMA bits coming from a reliable source which is using OSNMA. Doing this check, they can achieve some level of assurance on the signal authenticity without having to implement the complete OSNMA protocol [20], also known as Assisted NMA [21].

Additionally, the bits may be used by any other OSNMA library or GNSS-based application that needs to process navigation messages but does not have access to a GNSS receiver.

VI. Conclusion

In this paper, we presented the last updates on OSNMAlib, an open-source library to process the Galileo OSNMA protocol. Moreover, we discussed how distinct data sources (SBF, UBX, GNSS-SDR, galmon) can be added to OSNMA libraries, focusing on possible pitfalls and recommendations. Additionally, we presented the *osnmalib.eu* website, where real-time information of the OSNMA protocol can be consulted, alongside with the navigation data bits being processed by the library.

On the analysis of multiple GNSS data sources, we noticed that only Septentrio receivers provide a somewhat straightforward way of using their output for OSNMA processing. All the other sources considered lacked some key information, usually the time reference. We expect that, as OSNMA increases in popularity, more GNSS products will enable proper logging options for OSNMA applications.

With the exposure of the navigation data bits from the OSNMAlib website, we try, on the one hand, to become a source of GNSS data for applications, and, on the other hand, to help the developing and testing of OSNMA snapshot authenticated receivers.

Currently, *osnmalib.eu* only displays instantaneous data, which is refreshed every 30 seconds. An historical visualization of OSNMA data could be extremely useful for the GNSS community and be used for the analysis of OSNMA coverage, the cross-authentication algorithm, and numerous other applications.

REFERENCES

[1] A. Perrig, J. Tygar, A. Perrig, and J. Tygar, "TESLA broadcast authentication," *Secure Broadcast Communication: In Wired and Wireless Networks*, pp. 29–53, 2003.

[2] L. Scott, "Anti-spoofing & authenticated signal architectures for civil navigation systems," in *Proceedings of the 16th International Technical Meeting of the Satellite Division of The Institute of Navigation (ION GPS/GNSS 2003)*, 2003, pp. 1543–1552.

[3] T. E. Humphreys, "Detection strategy for cryptographic GNSS anti-spoofing," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 49, no. 2, pp. 1073–1090, 2013.

[4] I. Fernandez-Hernandez, V. Rijmen, G. Seco-Granados, J. Simon, I. Rodríguez, and J. D. Calle, "A navigation message authentication proposal for the Galileo open service," *NAVIGATION: Journal of the Institute of Navigation*, vol. 63, no. 1, pp. 85–102, 2016.

[5] M. Götzelmann, E. Köller, I. Viciano-Semper, D. Oskam, E. Gkougkas, and J. Simon, "Galileo open service navigation message authentication: Preparation phase and drivers for future service provision," *NAVIGATION: Journal of the Institute of Navigation*, vol. 70, no. 3, 2023.

[6] A. Galan, I. Fernandez-Hernandez, L. Cucchi, and G. Seco-Granados, "OSNMAlib: An Open Python Library for Galileo OSNMA," in *2022 10th Workshop on Satellite Navigation Technology (NAVITEC)*, 2022, pp. 1–12.

[7] A. Galan, I. Fernandez-Hernandez, G. Seco-Granados. (2024) OSNMAlib. GitHub repository. [Online]. Available: https://github.com/Algafix/OSNMA

[8] D. Estévez. (2024) galileo-osnma. GitHub repository. [Online]. Available: https://github.com/daniestevez/galileo-osnma/

[9] T. Hammarberg, J. M. V. García, J. N. Alanko, and M. Z. H. Bhuiyan, "FGI-OSNMA: An Open Source Implementation of Galileo's Open Service Navigation Message Authentication," in *Proceedings of the 36th International Technical Meeting of the Satellite Division of The Institute of Navigation (ION GNSS+ 2023)*, Denver, Colorado, September 2023, pp. 3774–3785.

[10] A. Galan and S. Fontfreda. (2024) OSNMAlib website. Accessed: February 8, 2024. [Online]. Available: https://osnmalib.eu/

[11] "European GNSS (Galileo) Open Service, Galileo OSNMA SIS ICD, Issue 1.1," European Union, Tech. Rep., Oct 2023.

[12] "European GNSS (Galileo) Open Service, Galileo OSNMA Receiver Guidelines, Issue 1.3," European Union, Tech. Rep., Jan 2024.

[13] "European GNSS (Galileo) Open Service, Signal-In-Space ICD, Issue 2.1," European Union, Tech. Rep., Nov 2023.

[14] C. Fernández-Prades. (2024) GNSS-SDR. CTTC. Open-source GNSS software-defined receiver. [Online]. Available: https://gnss-sdr.org

[15] Google. (2024) Protocol buffers - google developers. Accessed on: March 7, 2024. [Online]. Available: https://protobuf.dev/

[16] B. Hubert. (2024) Galmon network. GitHub repository. [Online]. Available: https://github.com/berthubert/galmon

[17] S. Damy, L. Cucchi, and M. Paonni, "Performance Assessment of Galileo OSNMA Data Retrieval Strategies," in *2022 10th Workshop on Satellite Navigation Technology (NAVITEC)*, 2022.

[18] A. Galan, I. Fernandez-Hernandez, W. De Wilde, S. Pollin, and G. Seco-Granados, "Improving galileo osnma time to first authenticated fix," *arXiv preprint arXiv:2403.14739*, 2024.

[19] Septentrio NV. (2024) mosaic-X5 GNSS receiver module. Accessed: February 28, 2024. [Online]. Available: https://www.septentrio.com/en/products/gps/gnss-receiver-modules/mosaic-x5

[20] H. Shahid, L. Canzian, C. Sarto, O. Pozzobon, J. Reyes-González, G. Seco-Granados, and J. A. López-Salcedo, "Spoofing Detection Performance of Snapshot OSNMA Under Time and Symbol Errors," in *2023 IEEE 98th Vehicular Technology Conference (VTC2023-Fall)*, 2023, pp. 1–7.

[21] C. O'Driscoll, J. Winkel, and I. Fernandez-Hernandez, "Assisted NMA proof of concept on Android smartphones," in *2023 IEEE/ION Position, Location and Navigation Symposium (PLANS)*. IEEE, 2023, pp. 559–569.