Improving OSNMAlib: New Formats, Features, and Monitoring Capabilities

Aleix Galan-Figueras, *Student Member, IEEE*, Cristian Iñiguez, Ignacio Fernandez-Hernandez, Sofie Pollin, *Senior Member, IEEE*, and Gonzalo Seco-Granados, *Fellow, IEEE*

***Abstract*—Galileo will declare Open Service Navigation Message Authentication (OSNMA), a civil Global Navigation Satellite System (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 article, 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, we describe how can they be integrated within an OSNMA library and what pitfalls to avoid. The new features include optimizations for data retrieval such as the use of dual frequency and Reed-Solomon encoding, which are evaluated in urban and open sky scenarios using real recorded data. The new JavaScript Object Notation (JSON) logging format aimed at researchers 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. In addition, the website also provides the I/NAV data bits to help snapshot receivers and other GNSS-based applications.**

***Index Terms*—Authentication, Galileo, global navigation satellite system, open service navigation message authentication (OSNMA), OSNMAlib.**

I. Introduction

GALILEO is the first Global Navigation Satellite System (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 Open Service Navigation Message Authentication (OSNMA) 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 timed efficient stream loss-tolerant authentication (TESLA) 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 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 the integration of OSNMA into their GNSS software-defined receiver.

In this article, we present the newly added input sources to OSNMAlib (Septentrio Binary Format (SBF), UBX, Software Defined Receiver (GNSS-SDR), galmon, and the Android GnnsLogger App) and give recommendations to other libraries willing to incorporate the same input sources. Then, we disseminate the new optimizations implemented in OSNMAlib, focusing on dual frequency and Reed-Solomon (RS) data recovery, and validate them using the time to first authenticated fix (TTFAF) metric with real data recordings in urban and open sky scenarios. Finally, we introduce the new logging features implemented in OSNMAlib and how we use them to build a

Received 5 November 2024; revised 22 February 2025; accepted 31 March 2025. Date of publication 8 April 2025; date of current version 8 May 2025. This work was supported by the Research Foundation Flanders (FWO) Frank de Winne PhD Fellowship, Project 1SH9424N. *(Corresponding author: Aleix Galan-Figueras.)*

Aleix Galan-Figueras and Sofie Pollin are with the Electrical Engineering Department, Katholieke Universiteit Leuven (KUL), 3000 Leuven, Belgium (e-mail: aleix.galan@kuleuven.be; sofie.pollin@kuleuven.be).

Cristian Iñiguez is with the Department of Telecommunication Engineering, Universitat Autònoma de Barcelona (UAB), 08193 Barcelona, Spain (e-mail: cristian.iniguez@autonoma.cat).

Ignacio Fernandez-Hernandez is with the Electrical Engineering Department, Katholieke Universiteit Leuven (KUL), 3000 Leuven, Belgium, and also with the European Commission, B-1049 Brussels, Belgium (e-mail: ignacio.fernandez-hernandez@ec.europa.eu).

Gonzalo Seco-Granados is with the Department of Telecommunication Engineering, Universitat Autònoma de Barcelona (UAB), 08193 Barcelona, Spain, and also with the Institute of Space Studies of Catalonia (IEEC), 08860 Castelldefels, Spain (e-mail: gonzalo.seco@uab.cat).

Digital Object Identifier 10.1109/JISPIN.2025.3558771

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 Galileo Interface Control Document (ICD) [14].

real-time OSNMA monitoring website [10], which includes the exposure of navigation message bits.

This article is a journal extension of [11]. We extend the original work by adding the GnnsLogger input (see Section III-E), and the optimizations dual frequency reception (see Section IV-C) and RS recovery (see Section IV-D). Then, we use real recordings to evaluate how much these optimizations improve the OSNMA performance in harsh environments (see Section V). We have also updated the JavaScript Object Notation (*JSON*) logging of OSNMAlib (see Section IV-A), and we distribute in a public endpoint the last processed subframes in *JSON* format (see Section VI-B).

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 b each that are grouped in a 30-s subframe. The nominal pages are divided into even and odd pages of 120 b 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 b on each nominal page. These bits are concatenated at the end of each subframe to create complete OSNMA structures. These structures are the Header and Root Key (HKROOT), with 120 b, which transmits the long-term authentication data to verify the keys; and the MAC and Key (MACK), with 480 b, 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 ICD [12] and in the receiver guidelines [13].

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 b of OSNMA message, and the Satellite Vehicle ID (SVID) to which the navigation data belongs. In addition, OSNMA needs to know the Galileo satellite

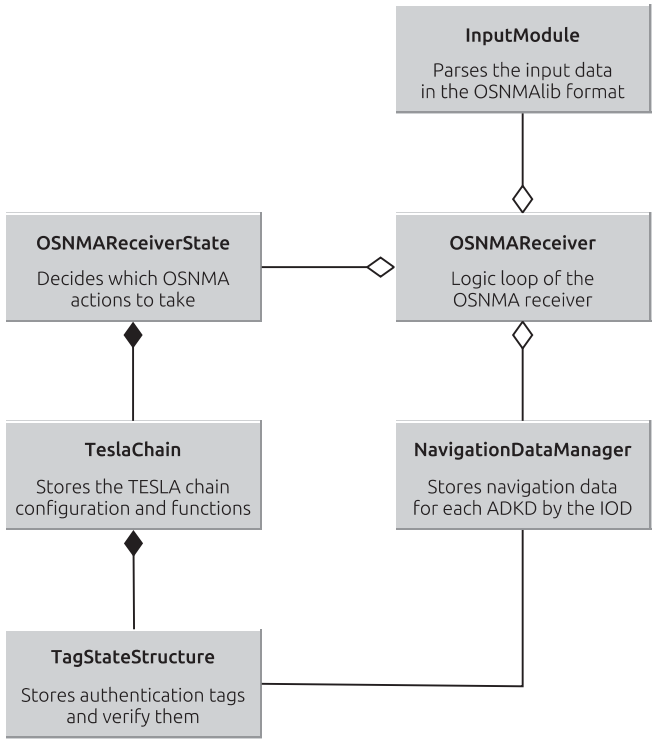


Fig. 2. Simplified UML diagram of OSNMAlib. The `OSNMAReceiver` is the main class and contains the `InputModule`, the `NavigationDataManager` to store the navigation data, and the `OSNMAReceiverState` class to process OSNMA. The `OSNMAReceiverState` creates the `TeslaChain` class when appropriate, which handles the verification of the tags.

time (GST) computed by the GNSS receiver to validate the time synchronization of the navigation data received and detect replay attacks. Moreover, in an offline scenario, accessing the receiver computed time is the only way to postprocess OSNMA.

In OSNMAlib, the input data is encapsulated in an object that contains the I/NAV message data from E1-B (240 b), 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 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 loops over the input data from an iterator and processes it: 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 subframes. 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 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 reuse 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 [13]. 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 b of an I/NAV navigation page, which results from the concatenation of the even and odd subpages 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 b 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 us to postprocess 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.

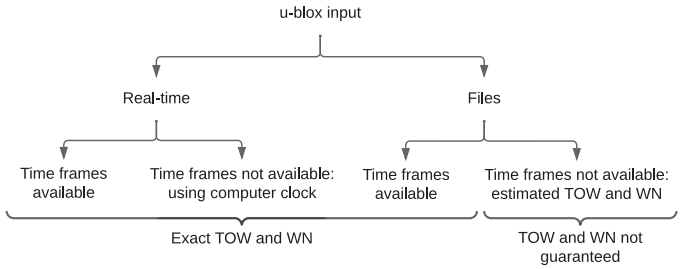


Fig. 3. OSNMAlib calculation of GST for the u-blox receiver input source depending on the available data.

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 WTs, 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.

The proposed strategy when postprocessing a file without time frames is based on the known sequence of WTs in a subframe, as defined in the Galileo ICD [14]. 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 subframe, 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 [14]

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

TABLE II
RELEVANT TTFAF VALUES FOR THE PROPOSED OPTIMIZATIONS IN THE SOFT URBAN SCENARIO

Optimization	Lowest (s)	Average (s)	P95 (s)
IOD SotA.	70.0	127.5	248.0
COP-IOD. Page proc. (A)	54.0	87.5	129.0
(A), and RS.	54.0	83.9	124.0
(A), and Dual-Freq.	54.0	82.1	120.0
(A), Dual-Freq, and RS.	54.0	81.6	119.0

TABLE III
RELEVANT TTFAF VALUES FOR THE PROPOSED OPTIMIZATIONS IN THE HARD URBAN SCENARIO

Optimization	Lowest (s)	Average (s)	P95 (s)
IOD SotA.	70.0	266.1	427.0
COP-IOD. Page proc. (A)	60.0	146.1	305.0
(A), and RS.	60.0	132.4	269.5
(A), and Dual-Freq.	60.0	114.1	232.0
(A), Dual-Freq, and RS.	60.0	111.5	221.5

TABLE IV
RELEVANT TTFAF VALUES FOR THE PROPOSED OPTIMIZATIONS IN THE OPEN SKY SCENARIO

Optimization	Lowest (s)	Average (s)	P95 (s)
IOD SotA.	70.0	84.5	98.0
COP-IOD. Page proc. (A)	60.0	68.8	87.0
(A), and RS.	60.0	68.8	87.0
(A), and Dual-Freq.	60.0	68.8	87.0
(A), Dual-Freq, and RS.	60.0	68.8	87.0

C. GNSS-SDR

GNSS-SDR [15] is an open-source GNSS software-defined receiver written in C++ that supports multiple constellations, including Galileo, and real-time kinematics 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 postprocess), 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* [16] 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 b 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 WT 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* [17] 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. In addition, 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 Section 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

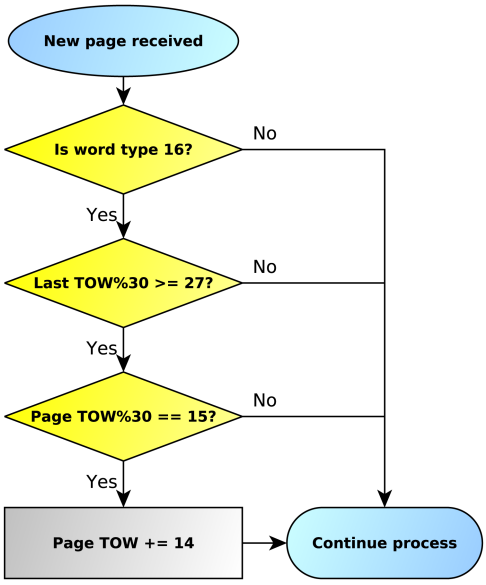


Fig. 4. Fix at library level to correct the assignation by galmon of the TOW for the WT 16.

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-b 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 b can be set to 0.

There are two considerations when integrating the gallon network with an OSNMA library. The first is that, at the time of writing this article, the algorithm used by the network to estimate the TOW of pages coming from u-blox receivers does not account for the latest changes in the Galileo ICD with regard to the position of WT 16. Currently, WT 16 is transmitted on pages eight and 15, but previously, it was only transmitted on 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 WT 16 is transmitted in the new position with the wrong TOW value.

The second consideration is that the messages in the network do not necessarily follow a chronological order, and some receivers send messages that are 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.

E. Android GnssLogger App

The Android GnssLogger App [18] is an official app developed by Google to register and analyze location data from GNSS or other sensors. The logs are sentences in American Standard Code for Information Interchange (ASCII) format and contain information from the multiple sensors (accelerometer, gyroscope, and magnetometer), the status of the GNSS front end (automatic gain control and signal power), the position coordinates, and the navigation message bits. The logged data is exactly the same as the one available from the Android operating system API. In other words, it allows us to use all the location information available in an Android smartphone without developing a dedicated application.

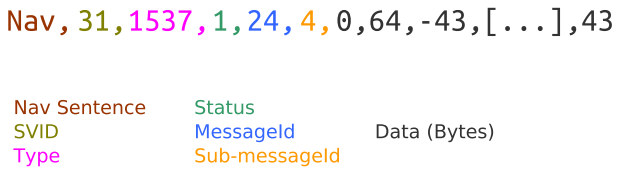


Fig. 5. Parsing of the *Nav* logging sentence containing the navigation data bits from the Android application *GnssLogger*.

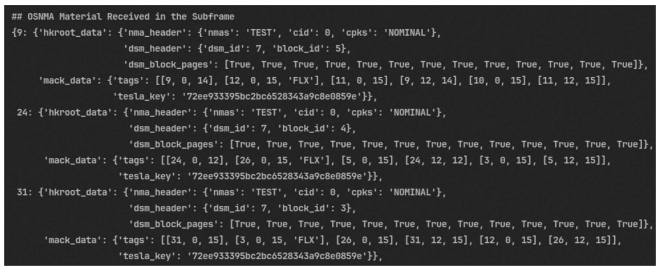
The most interesting material to integrate it with OSNMA is contained in the *Nav* sentence. This sentence, dissected in Fig. 5, contains the following values.

1) *SVID:* The SVID of the transmitting satellite. For Galileo, it ranges from 1 to 36.
2) *Type:* Describes the type of navigation message. For OSNMA, the relevant message is the Galileo I/NAV, which has a value of 1537.
3) *Status:* Contains the integrity status of the navigation data, parity passed is 1.
4) *MessageId:* For Galileo I/NAV this refers to the subframe ID, and has a value between 1 and 24.
5) *Sub-messageId:* For Galileo I/NAV this refers to the WT contained in the page.
6) *Data(Bytes):* Contains the full page (concatenated even and odd) without the tail bits, for a total of 228 b (padded in 29 B).

A vital piece of information for OSNMA is missing from the *Nav* sentence: the time. Curiously, all the other messages contain the time information in a field called *utcTimeMillis*, but not *Nav*. Thus, we can use these other messages to keep track of the time.

However, another problem arises when using the additional messages. The time we are interested in is the GNSS time at the transmission of the page, and the time reported by the other messages is when their measurements were taken. Moreover, the sensor and GNSS related messages are not chronologically sorted in the log file, and the time reported between them is OFF by a few seconds. Therefore, attempting to keep time simply by reading the last *utcTimeMillis* read will not suffice.

For the OSNMAlib implementation, we have decided only to keep time using the GNSS-related sentences *Raw* and *AGC*. These two sentences have proven to be more reliable when compared with the known time reported in some navigation data pages. The only correction needed is to subtract two seconds from their time to translate the navigation data reception time to transmission time.



```
## OSNMA Material Received in the Subframe
{9: {'hkroot_data': {'nma_header': {'nmas': 'TEST', 'cid': 0, 'cpks': 'NOMINAL'},
                     'dsm_header': {'dsm_id': 7, 'block_id': 5},
                     'dsm_block_pages': [True, True, True, True, True, True, True, True, True, True, True, True, True, True, True]},
      'mack_data': {'tags': [[9, 0, 14], [12, 0, 15, 'FLX'], [11, 0, 15], [9, 12, 14], [10, 0, 15], [11, 12, 15]],
                    'tesla_key': '72ee933395bc2bc6528343a9c8e0859e'}},
 24: {'hkroot_data': {'nma_header': {'nmas': 'TEST', 'cid': 0, 'cpks': 'NOMINAL'},
                      'dsm_header': {'dsm_id': 7, 'block_id': 4},
                      'dsm_block_pages': [True, True, True, True, True, True, True, True, True, True, True, True, True, True, True]},
      'mack_data': {'tags': [[24, 0, 12], [26, 0, 15, 'FLX'], [5, 0, 15], [24, 12, 12], [3, 0, 15], [5, 12, 15]],
                    'tesla_key': '72ee933395bc2bc6528343a9c8e0859e'}},
 31: {'hkroot_data': {'nma_header': {'nmas': 'TEST', 'cid': 0, 'cpks': 'NOMINAL'},
                      'dsm_header': {'dsm_id': 7, 'block_id': 3},
                      'dsm_block_pages': [True, True, True, True, True, True, True, True, True, True, True, True, True, True, True]},
      'mack_data': {'tags': [[31, 0, 15], [3, 0, 15, 'FLX'], [26, 0, 15], [31, 12, 15], [12, 0, 15], [26, 12, 15]],
                    'tesla_key': '72ee933395bc2bc6528343a9c8e0859e'}},
```


Fig. 6. Example of the OSNMA material received section of the OSNMAlib status *JSON* logging in the console. The *JSON* logging consists of 5 sections: OSNMA status, OSNMA material received, navigation data received, verified OSNMA material, and authenticated navigation data. The same information is logged in pure *JSON* to a file.

Finally, a last remark is that the Galileo I/NAV message is transmitted in two signals, E1B and E5b-I, with the OSNMA bits only present in E1B, and in the current GnssLogger format, there is no direct way of distinguishing between the messages coming from the two signals. Attempting to parse the OSNMA bits in a message coming from E5b-I will have undefined behavior because the same bit range is set as reserved on that signal. A discrimination method could be to use the one-second offset between the signals in the transmission of the two-second I/NAV pages, or to compare the page received with the nominal page order listed in the Galileo ICD (albeit it is only indicative). A better approach would be, however, for GnssLogger to indicate the signal of origin in the sentence (for example, in the *Type* field).

IV. New Features

A. Pure JSON 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 OSNMA relevant metrics in mind, we developed a new logging stream that reports at the end of every subframe (i.e., every 30 s) on the state of OSNMAlib. This logging option presents in *JSON* format the following information per subframe: OSNMA status, OSNMA material received, navigation data received, verified OSNMA material, and authenticated navigation data. Fig. 6 shows an example of how this information is displayed in the console in a readable way, but it is also saved into a file 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.

To define the contents of the *JSON* status file in a standardized way, OSNMAlib also provides a *json-schema*. The *json-schema* describes which fields to expect in every object and in which format. For example, and following the example on Fig. 6, it defines that for each satellite transmitting OSNMA there is an object, which contains the *HKROOT* and *mack* data, it also defines the meaning of the numbers in the *tags* array, and the possible values for the enumerations such as *nmas* or *cpks*.

B. OSNMA 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 subframes. This strategy was first described in [19] and greatly improves the TTFAF and continuous authentication of navigation data.

This approach allows the extraction of each authentication tag individually instead of parsing a fully received subframe. In this way, even if only two pages out of the 15 that form the subframe 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 subframe'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 subframes. 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 subframe was also transmitted in the previous. This assumption allows us to get TTFAF in hot start as low as 44 s [20].

C. Galileo Dual-Frequency Data Reception

The I/NAV message authenticated by OSNMA is transmitted in two signals: E1-B (1575.42 MHz) and E5b-I (1207.14 MHz). The reception and process of the E1-B signal is mandatory to use OSNMA, since it is where the protocol bits are transmitted. Nonetheless, a dual-frequency receiver can also use E5b-I to receive the navigation data for a more resilient performance.

The navigation data words are transmitted in a different order in the E1-B and E5b-I signals, and this allows us to receive faster all the words needed to authenticate the satellite ephemeris (WTs 1 to 5). Fig. 7 shows the nominal location of the ephemeris words inside the subframe for both signals. A complete set of words 1 to 5 can be received at the beginning and at the end of the subframe, and they can be combined if partially received.

In OSNMAlib, we implement the dual-frequency reception as an option to the input sources. By default, the navigation data pages introduced to OSNMAlib are assumed to come from the E1-B signal and it can be specified that they come from E5b-I. Since the word bits position inside the page is identical between both bands, the only practical difference in the processing of the pages coming from different signals is the extraction or not of the OSNMA bits.

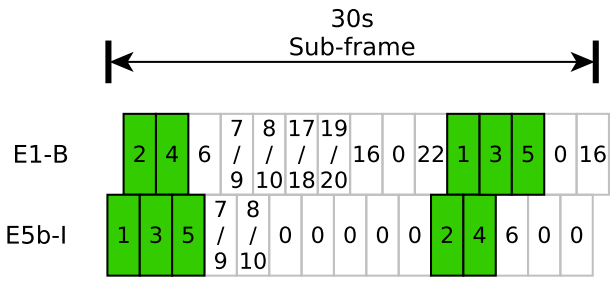


Fig. 7. Pages containing WTs needed for the ADKD0 authentication (WT 1–5) are marked in green. By using dual frequency, two sets of these words are transmitted every subframe.

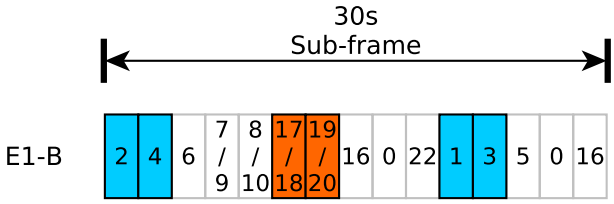


Fig. 8. WTs protected by RS (WT 1–4) are marked in blue, and the RS parity pages that act as wildcards for any of the protected ones are marked as orange.

D. Galileo RS Page Recovery

As part of the latest improvements to the Galileo I/NAV message, Galileo introduced an outer forward error correction based on RS, which is applied to WTs 1 to 4. The recovery of those words is a great way of improving the performance of OSNMA in challenging scenarios where some of them may get lost, even if WT5, which is also needed for ADKD0 authentication, is not protected.

The RS encoding is constructed over a Galois Field of order 256, defined by the primitive polynomial

$$p(x) = x^8 + x^4 + x^3 + x^2 + 1 \tag{1}$$

which has a decimal representation of 285. The vector length is 255 octets, but the shortened version used by Galileo has a code length of 118 octets: 58 octets for the information vector and 60 octets for the parity vector.

The information vector is constructed by concatenating WT 1 to 4, with some octets discarded. The parity vector is transmitted in WT 17 to 20, and in each subframe, only 2 parity pages are transmitted in the E1-B signal in alternating order. The parity vector pages contain a reference to the navigation data IOD transmitted in WT 1 to 4 to facilitate the linking. Fig. 8 indicates in blue the information pages and in orange the parity pages in one subframe.

The parity pages act as wildcard pages: the receiver needs 4 different pages in total (coming either from the information vector or the parity vector) to recover WT 1 to 4. In a practical example, a receiver could miss the information words 2 and 4 transmitted at the beginning of the subframe, but recover the parity words 17 and 19 together with the information words 1 and 3 transmitted later. Then, by the RS decoding it could recover WT 2 and 4. In a more drastic example, it is also possible to recover WT 1–4 from WT 17–20.

From an implementation point of view, the majority of Python libraries that implement RS need to be tweaked to adjust to the Galileo version. While the primitive polynomial is quite common, the first consecutive root (FCR) default value, which defines in which iteration the encoding starts, varies between implementations. The FCR value in MATLAB implementations and the Galileo format is set to 1, but in the majority of Python libraries, it defaults to 0. The information and parity vector order may also need to be reversed when using the common Python RS libraries. More information about the RS encoding and examples can be found in the Galileo Signal-in-Space Interface Control Document [14].

V. Optimizations Impact in Real Scenarios

To evaluate the impact of the optimizations implemented in OSNMAlib in real scenarios, we have used recorded data in soft urban, hard urban, and open-sky environments, which is available in [21]. The log files are continuous recordings between 30 and 40 min of two walks in Brussels and a static scenario on the roof of Septentrio headquarters in Leuven.

The optimizations are evaluated using the TTFAF metric in the *hot start* operation mode. The *hot start* mode is the most common start mode in OSNMA and assumes that the receiver has already in its possession the root key (or an intermediate authenticated key) of the TESLA chain. The chain is expected to last for several months; thus, the receiver can store the key in memory and use it when powering up.

The TTFAF requires the receiver to have authenticated ephemeris for a minimum of four satellites. The ephemeris is authenticated using the ADKD0 tag of OSNMA, so the requirement translates to being able to authenticate an ADKD0 tag for a minimum of four satellites.

To simulate the multiple TTFAF values, we start to process the log files with OSNMAlib one second later every time. This strategy emulates a receiver starting to decode navigation data at a different moment each time, and for each log file, we can obtain around 2000 points with varying geometry.

For the configuration options compared, we have considered the following.

1) A basic OSNMA implementation that links navigation data and tags based on the IOD.
2) The best case of [20], which includes page-level processing, the COP-IOD optimization, and a reduced time synchronization of 17 s (briefly discussed in Section IV-B and in more depth in the cited manuscript).
3) The best case of [20] and adding RS recovery for the navigation words 1 to 4 as described in Section IV-D.
4) The best case of [20] and adding dual frequency reception for the I/NAV message in the signal E5b-I as described in Section IV-C.
5) The best case of [20] and adding RS recovery and dual frequency reception.

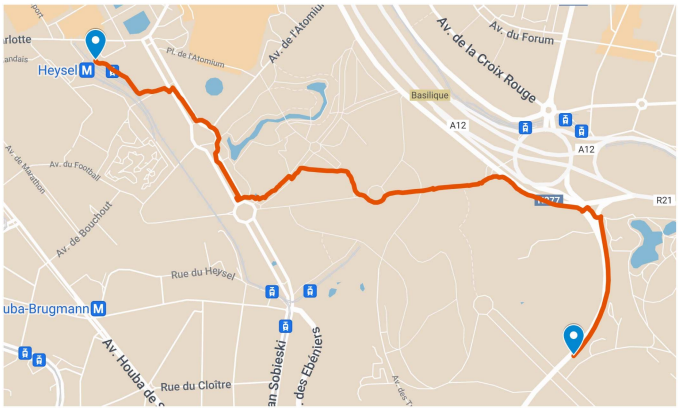


Fig. 9. Trajectory of the soft urban scenario recording on 3 December 2023 in Brussels, Belgium.

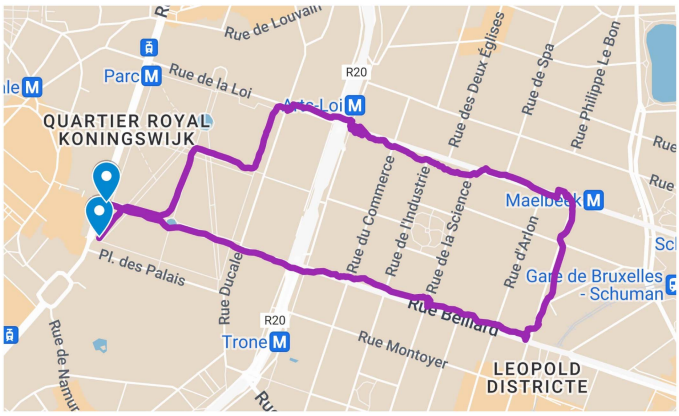


Fig. 10. Trajectory of the hard urban scenario recording on 3 December 2023 in Brussels, Belgium.

A. Scenarios

The scenarios were recorded during December 2023, but the OSNMA configuration at that time was the same as the one transmitted at the time of publishing this manuscript. In any case, the results show the relative improvement when enabling the optimizations in OSNMAlib.

The soft urban scenario recording is a walk around the Atomium and Laeken Park in Brussels, Belgium, on 3 December 2023, from 11:03:24 to 11:43:53 UTC. The trajectory is shown in Fig. 9; there are no big buildings around, and the satellite visibility is limited only due to the trees.

The hard urban scenario recording is a walk in the European District of Brussels, Belgium, on 3 December 2023, from 09:50:00 to 10:22:30 UTC. The trajectory is shown in Fig. 10; the satellite visibility is strongly limited due to the urban canyon environment with tall buildings, and there is a lot of fading and intermittent reception.

The open sky scenario is a 60-min static recording from the Septentrio offices in Leuven, Belgium, on 20 December 2023, from 15:00:00 to 16:00:00 UTC. The visibility is excellent during the whole recording, with satellites in view from low elevation. The number of lost pages and satellite fading is very low since there are no objects blocking the sight.

B. Results

The TTFAF results in the form of a cumulative distribution function (CDF) are shown in Fig. 11. Unsurprisingly, the OSNMAlib configuration with all optimizations enabled performs the best in all scenarios.

The optimization that brings more relative improvement is the use of COP-IOD and page-level processing. The page-level processing allows not to discard correctly received pages because others on the same subframe were lost, which is a huge improvement in harsh environments such as the urban ones. The COP-IOD optimizations allow the linking of partially received navigation data with authentication tags, but require very good visibility of satellites. Hence, the COP-IOD part works best in open-sky scenarios. The nuances of these two optimizations are further discussed in [20].

On top of this case, the dual frequency optimization improves more the TTFAF than the RS recovery optimization. These results are to be expected if we analyze how many words can be recovered with each optimization. By receiving the I/NAV message in the E5b-I signal, we are receiving an extra set of WT 1–5 for each subframe, and these are received time-correlated with the same WTs in E1-B. Thus, the receiver can quickly collect the full set of WT in a few seconds of good visibility, twice per subframe.

The RS recovery includes 2 extra WT that can be used as a wildcard for any of the WT 1–4. While it is a great addition and arguably more flexible than the fixed words obtained by using E5b-I, it has less amount of extra information. Moreover, it protects only WT 1–4, so the receiver still needs to receive the single time transmitted WT5 each subframe. However, one clear advantage is that a receiver can use the RS optimization without any hardware change, unlike the dual frequency optimization.

Finally, it is worth mentioning that these optimizations work better when the scenario is more harsh. The improvement in the hard urban scenario is bigger than in the soft urban scenario because more pages are lost, and the use of dual frequency or RS becomes crucial to have a faster fix. Going to the extreme of the open sky scenario, where no pages are lost, neither the RS nor the dual frequency optimization improve the TTFAF values. It is true that the receiver may start mid subframe and miss some pages so it would benefit from the redundancy introduce by dual-frequency and RS, but because in the next subframe all pages are received the COP-IOD optimization alone is already capable of recovering them.

VI. OSNMA Status Website

A. OSNMAlib Dashboard

Using the new subframe status logging of OSNMAlib described in Section IV-A, 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 [22]

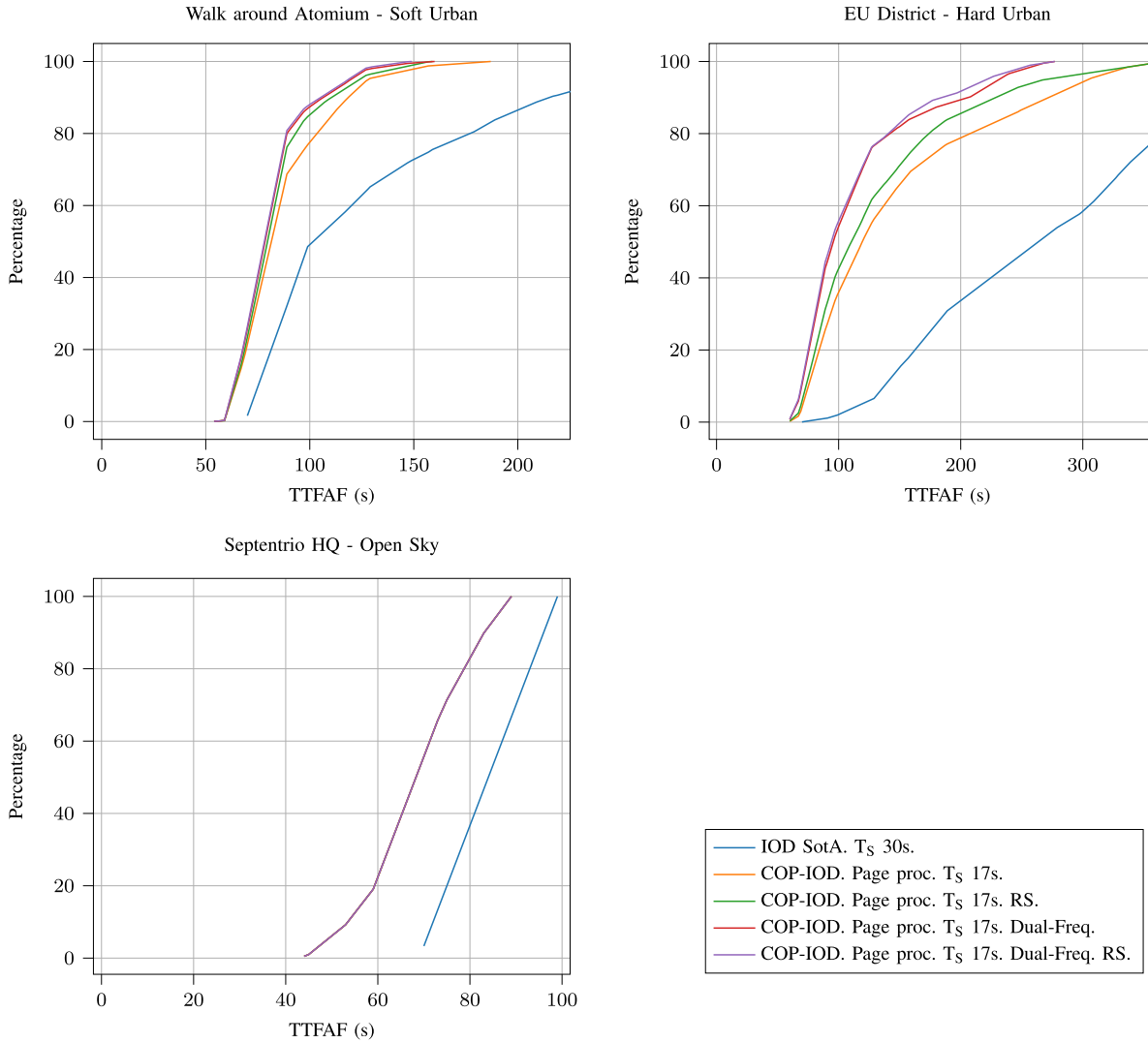


Fig. 11. CDFs of the TTFAF values with different optimizations enabled in multiple environments. The COP-IOD with page processing optimization is the best case of [20], and brings the most relative improvement. The use of dual frequency improves more the TTFAF than the use of RS, but requires hardware changes on the receiver. The use of all the optimizations lies always the best results.

located at KU Leuven, Belgium, and the other processing global aggregated data from the galmon network.

After each subframe, 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. 12. The navigation data WTs 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. 13. Each color represents a tag type in the same sequence mentioned above. The first column contains the 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.

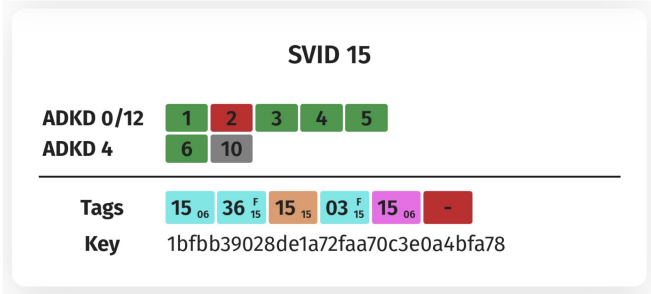


Fig. 12. Data blocks obtained from satellite 15 in the last subframe as displayed in the OSNMAlib website.

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

Fig. 13. Navigation data authentication status for satellite 15 as displayed in the OSNMAlib website.

B. OSNMAlib Status JSON Endpoint

The *JSON* objects used to populate the website dashboard are also provided to the users through the `/septentrio/subframe_json` and `/galmon/subframe_json` endpoints. In this way, anyone can access to the OSNMA information from the receivers in a structured way and choose to process it at will. Also, the *JSON* object contains much more information than the one displayed in the dashboard.

The endpoints return the OSNMAlib information for the last 12 subframes. The decision of saving 12 subframes is to always have in memory an ADKD12 tag and the applicable TESLA key (transmitted with an additional delay of 10 subframes).

In the website, we also provide direct links to the *json-schema* that describes the *JSON* object returned by the endpoint. The schema is provided in an *HTML* version, which is targeted at the user trying to figure out what information is provided, and also in *JSON* version so it can be injected as a reference to development environments willing to process the endpoint objects.

This OSNMAlib subframe information could be used by other researchers that want to analyze the OSNMA protocol but do not have access to a receiver to process the data in real time. Also, it could be used to retrieve the TESLA keys needed to use the Galileo SAS protocol.

C. 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 subframe, identified by the WN and TOW, and organized per satellite. Each satellite contains 15 pages of 240 b with the navigation message in hexadecimal. If a page is lost the value for the page is `null`. Fig. 14 shows how the *JSON* structure looks like. These endpoints also return the last 12 subframes bits from the selected receiver.

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 [23], also known as Assisted NMA [24].

In addition, 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.

Fig. 14. Example of the I/NAV bits as retrieved from the *osnmalib.eu* website. The *JSON* structure contains the last 12 subframes received by OSNMAlib, and each subframe contains the WN, the TOW and the bits from each satellite organized in 240 b pages.

VII. Conclusion

In this article, we presented the last updates on OSNMAlib, an open-source library to process the Galileo OSNMA protocol. We discussed how distinct data sources (SBF, UBX, GNSS-SDR, gallon, and GnssLogger) can be added to OSNMA libraries, focusing on possible pitfalls and recommendations. In addition, we analyzed using real data how processing data from the E5b-I signal and implementing the RS recovery can greatly increase the TTFAF in harsh environments. Finally, 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. Moreover, by providing the subframe information in *JSON* format we also enable other applications that need access to verifies information transmitted by OSNMA such as the Galileo SAS protocol.

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 Commun.: Wired Wireless Netw.*, pp. 29–53, 2003.

[2] L. Scott, "Anti-spoofing & authenticated signal architectures for civil navigation systems," in *Proc. 16th Int. Tech. Meeting Satell. Division Inst. Navigat.*, 2003, pp. 1543–1552.

[3] T. E. Humphreys, "Detection strategy for cryptographic GNSS anti-spoofing," *IEEE Trans. Aerosp. Electron. Syst.*, vol. 49, no. 2, pp. 1073–1090, Apr. 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," *Navigat.: J. Inst. Navigat.*, 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: J. Inst. Navigat.*, vol. 70, no. 3, 2023. [Online]. Available: https://navi.ion.org/content/70/3/navi.572

[6] A. Galan, I. Fernandez-Hernandez, L. Cucchi, and G. Seco-Granados, "OSNMAlib: An Open Python Library for Galileo OSNMA," in *Proc. 10th Workshop Satell. Navigat. Technol.*, 2022, pp. 1–12.

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

[8] D. Estévez, "Galileo-osnma. GitHub repository," 2024. [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 *Proc. 36th Int. Tech. Meeting Satell. Division Inst. Navigat.*, Denver, Colorado, Sep. 2023, pp. 3774–3785.

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

[11] A. Galan, C. Iñiguez, I. Fernandez-Hernandez, S. Pollin, and G. Seco-Granados, "OSNMAlib improvements and real-time monitoring of Galileo OSNMA," in *Proc. Int. Conf. Localization GNSS*, 2024, pp. 1–7.

[12] "GNSS European (Galileo) Open service, Galileo OSNMA SIS ICD, issue 1.1," European Union, Maastricht, The Netherlands, Tech. Rep., Oct. 2023.

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

[14] "GNSS European (Galileo) Open service, signal-in-space ICD, issue 2.1," European Union, Maastricht, The Netherlands, Tech. Rep., Nov. 2023.

[15] C. Fernández-Prades, "GNSS-SDR. CTTC. open-source GNSS software-defined receiver," 2024. [Online]. Available: https://gnss-sdr.org

[16] Google, "Protocol Buffers - Google Developers," 2024. Accessed on: Mar. 7, 2024. [Online]. Available: https://protobuf.dev/

[17] B. Hubert, "Galmon network. GitHub repository," 2024. [Online]. Available: https://github.com/berthubert/galmon

[18] Google, "GnssLogger app. (version 3.1.0)," 2025. [Online]. Available: https://play.google.com/store/apps/details?id=com.google.android.apps.location.gps.gnsslogger

[19] S. Damy, L. Cucchi, and M. Paonni, "Impact of OSNMA configurations, operations and user's strategies on receiver performances," in *Proc. 35th Int. Tech. Meeting Satellite Division Inst. Navigation (ION GNSS+ 2022)*, Denver, Colorado, Sep. 2022, pp. 820–827, doi: 10.33012/2022.18302.

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

[21] A. Galan, I. Fernandez-Hernandez, and W. De Wilde, "GNSS recordings for Galileo OSNMA evaluation," *IEEE Dataport*, Mar. 13, 2024, doi: 10.21227/a0nm-kn45.

[22] N.V. Septentrio, "mosaic-X5 GNSS receiver module," 2024. Accessed: Feb. 28, 2024. [Online]. Available: https://www.septentrio.com/en/products/gps/gnss-receiver-modules/mosaic-x5

[23] H. Shahid et al., "Spoofing detection performance of snapshot OSNMA under time and symbol errors," in *Proc. IEEE 98th Veh. Technol. Conf.*, 2023, pp. 1–7.

[24] C. O'Driscoll, J. Winkel, and I. Fernandez-Hernandez, "Assisted NMA proof of concept on android smartphones," in *Proc. IEEE/ION Position, Location Navigat. Symp.*, 2023, pp. 559–569.

Aleix Galan-Figueras (Student Member, IEEE) received the B.Sc. degree in computer engineering and telecommunication systems engineering from the Universitat Autonoma de Barcelona (UAB), Bellaterra, Spain, in 2020, the M.Sc. degree in cybersecurity from the Universitat Politècnica de Catalunya (UPC), Barcelona, Spain, in 2022. He is currently working toward the Ph.D. degree in GNSS security and resilience with the WaveCoRE Research Group, ESAT, Katholieke Universiteit Leuven (KU Leuven), Leuven, Belgium.

In 2022, he undergo an Erasmus exchange with KU Leuven and Septentrio NV, Leuven, where he worked on his master's thesis. During his master's, he worked on a European Commission funded project with UAB to develop an open-source library for the Galileo OSNMA protocol. Then, he worked for two years in the industry with Septentrio NV on the topics of GNSS spoofing detection and Software Defined Radio devices.

Mr. Galan-Figueras was the recipient of the Ph.D. fellowship from FWO in 2023.

Cristian Iñiguez received the B.Sc. degree in computer science and telecommunications systems from the Universitat Autònoma de Barcelona, Bellaterra, Spain, in 2024. He is currently working toward the M.Sc. degree in advanced telecommunication technologies with the Universitat Politècnica de Catalunya, Barcelona, Spain.

He is currently a Research Engineer with the Barcelona Supercomputing Center, Barcelona, Spain. His research interests include high-performance computing, vector architectures, and parallel programming, with applications in genomics, sequence alignment, and signal processing.

Ignacio Fernandez-Hernandez received the Ph.D. degree in electronic systems from Aalborg University, Aalborg, Denmark, in 2015.

He is currently with the European Commission, Brussels, Belgium, where has led the design and development of Galileo high accuracy and authentication services over the last years. He also chairs the EU-US Resilience and EU Authentication and High Accuracy Working Groups. He is an Engineer from ICAI, Madrid, Spain.

Sofie Pollin (Senior Member, IEEE) received the engineering science degree and the Ph.D. (hons.) degree from KU Leuven, Leuven, in 2002 and 2006, respectively.

She is currently a Full Professor with KU Leuven focusing on wireless communication systems. She had a research position with UC Berkeley from 2006 to 2008, as a BAEF and Marie Curie Fellow. From 2008 to 2012, she was a Senior Researcher at imec, where she is currently still a Principal Member of technical staff. Her research centers around wireless networks that require networks that are ever more dense, heterogeneous, battery-powered, and spectrum-constrained. Her research interests include cell-free networks, integrated communication and sensing, and nonterrestrial networks.

Dr. Pollin is a Member of the Executive Editorial Committee for IEEE Transactions on Wireless Communications and an Associate Editor for IEEE Transactions on Mobile Computing. She is the publication and special issue Officer for the Aerial Communications Emerging Technology Initiative (AC-ETI). She was a TPC Co-Chair of the 4th and 5th IEEE Joint Communication and Sensing (JC&S) Symposium and of the 2024 EuCNC, symposium Co-Chair of Globecom 2021 (Communication Theory), Globecom 2022 (SAC AC), ICC 2024 (Communication Theory), PIMRC 2024, WCNC 2022, DySPAN 2015, and was involved in the organization of ICC 2020, DySPAN 2017, ISWCS 2015, CCNC 2016, and ACM Mobicom 2023. She was the recipient of ICC 2024 and EuCNC 2024 best paper awards.

Gonzalo Seco-Granados (Fellow, IEEE) received the Ph.D. degree in telecommunications engineering from the Univ. Politècnica de Catalunya, Barcelona, Spain, in 2000 and the master's degree in business administration from the IESE Business School, Barcelona, Spain, in 2002.

From 2002 to 2005, he was a member of the European Space Agency, Paris, France, where he was involved in the design of the Galileo system. He is currently a Professor with the Department of Telecommunication, Univ. Autònoma de Barcelona, Bellaterra, Spain, and with the Institute of Spatial Studies of Catalonia, and co-founder of Loctio. His research interests include GNSS, and 5G/6G localization and sensing.

Dr. Seco-Granados has been the President of the Spanish Chapter of the IEEE Aerospace and Electronic System Society, since 2019. He was the recipient of the 2021 IEEE Signal Processing Society's Best Paper Award.