FGI-OSNMA: An Open Source Implementation of Galileo's Open Service Navigation Message Authentication

Toni Hammarberg, José M. Vallet García, Jarno N. Alanko, and M. Zahidul H. Bhuiyan
Finnish Geospatial Research Institute

BIOGRAPHIES

Toni Hammarberg is a research scientist at the Finnish Geospatial Research Institute, where he has been working since 2017. Previously he has worked on Simultaneous Localization and Mapping techniques, and other sensor-based localization methods. More recently he has worked on GNSS signal monitoring and authentication. He is also a PhD student at the Department of Computer Science of University of Helsinki. His research interests include robust and resilient navigation.

José M. Vallet García obtained a M.Sc. in Electronics and Automation Technology from the Universidad Carlos III de Madrid, and a D.Sc in Automation Technology and Control Engineering from Aalto University. At the moment he is with the Finnish Geospatial Research Institute working with GNSS positioning and related services.

Jarno N. Alanko obtained his Ph.D. in computer science and bioinformatics in 2020 from the University of Helsinki. Since then, he has worked as a post-doc in Dalhousie University and the University of Helsinki, and developed software for the National Land Survey of Finland.

M. Zahidul H. Bhuiyan is a Research Professor at the Department of Navigation and Positioning in Finnish Geospatial Research Institute. He is also serving as an 'Adjunct Professor' in Tampere University. His main research interests include multi-GNSS receiver development, PNT robustness and resilience, seamless positioning, LEO-PNT user receiver development, etc. He has been also working as a Technical Expert for the EU Agency for the Space Program (EUSPA) in H2020/Horizon Europe project reviewing and proposal evaluation.

ABSTRACT

The European Global Navigation Satellite System (GNSS) Galileo is launching the Open Service Navigation Message Authentication (OSNMA) to enable navigation message authentication for all users, and therefore increasing the resiliency against spoofing. The Finnish Geospatial Research Institute (FGI) has developed an open source implementation of Galileo's OSNMA, henceforth known as FGI-OSNMA. FGI-OSNMA is a Python library functioning as a OSNMA computation engine with special emphasis put into its modularity, usability in real time, and integrability as a library in third party applications. In particular, the library is being integrated to the software receiver FGI-GSRx and the GNSS situational awareness service GNSS-Finland. In addition to this, our software package contains useful tools, such as scripts to compute and visualize key performance indicators (KPIs) related to authentication, and a filter to remove unauthenticated messages from RINEX navigation and observables files. This paper presents an overview of the features of FGI-OSNMA, followed by description of the architecture and the rationale behind the design. Finally, the paper concludes by demonstrating practical examples and real-world applications of the library.

I. INTRODUCTION

Global Navigation Satellite Systems (GNSS) have become a critical component in many systems and applications. Given the importance of the GNSS service and the observed increasing trends of intentional interference, its security has become a key aspect of its development. One particular security concern is that of *spoofing*, where an attacker transmits fake GNSS signals in an attempt to influence the position, velocity and time (PVT) solution of the receiver. Such attacks can have a significant impact in real-world applications. In Galileo Open Service Navigation Message Authentication (OSNMA), this problem is addressed by transmitting cryptographic data in the Signal-In-Space (SIS), which allows the receiver to verify that the received navigation message is unmodified and authentic. While cryptography and authentication are widespread, authentication of satellite data poses unique challenges, and Galileo OSNMA is the first to offer this service in the civilian sector. Although the use of OSNMA does not address all possible types of attacks Motallebighomi et al. (2022), the authentication that it enables significantly increases the security and resilience of a positioning service.

The Finnish Geospatial Research Institute (FGI) has made an OSNMA implementation called FGI-OSNMA within the context of the Horizon 2020 funded ESRIUM project ESRIUM (2023). This project aims to create road wear-maps with accurate

information about the position and shape of road damage, and provide drivers and autonomous vehicles instructions to avoid the damaged areas in the form of route recommendations (e.g. change of lanes and driving offsets with respect to the lane center). The ultimate goal is to increase the safety and reduce the road maintenance costs. In this project we use OSNMA to increase the security and robustness of the system. Understanding the practical characteristics of OSNMA was important in ESRIUM, and a flexible OSNMA implementation was needed to perform this analysis. FGI-OSNMA was used to investigate these performance characteristics of OSNMA in Hammarberg et al. (2023).

FGI-OSNMA is a flexible and open source OSNMA implementation written in Python. Similar to other open source projects of FGI, such as HASlib and FGI-GSRx Horst et al. (2022); Borre et al. (2022), it is published under the Github page of National Land Survey of Finland (NLS) FGI (2023). FGI-OSNMA supports all of the main features of OSNMA, which include, for example, self and cross-authentication of satellites, support for all of the different Authentication Data and Key Delay (ADKD) types, public-key renewal, and support for the different initialization paradigms (cold, warm, and hot start). In addition to implementing all of the core OSNMA features, FGI-OSNMA puts special emphasis on its software architecture with the goal of making it easily extendable and easy to integrate into other software projects. To highlight the ease of integration, we note that FGI-OSNMA is being integrated in two important software projects of FGI: the software receiver FGI-GSRx Bhuiyan et al. (2022) and the GNSS monitoring service GNSS-Finland (2023). Our software (SW) package also includes tools that use our OSNMA library for specific useful purposes. Examples of this include a tool for computing and visualizing authentication related key performance indicators (KPIs), or a filter for removing unauthenticated navigation messages from RINEX navigation files and optionally their associated measurements from RINEX observables files.

The Galileo OSNMA is currently in public observation test phase. Despite this, there are already at least two open source implementation of OSNMA Galan et al. (2022); Estevez (2022) in addition to FGI-OSNMA. Each of these OSNMA projects have different scope and goals, with FGI-OSNMA putting emphasis on modularity and integrability.

European Union Space Program Agency (EUSPA) has published so-called *test vectors* to validate the correctness of OSNMA implementations. The correctness of FGI-OSNMA has been validated by using these test vectors, and in addition to this, we have made extensive comparisons against other available OSNMA implementations, such as OSNMAlib Galan et al. (2022) and the Septentrio receivers' OSNMA implementation.

The rest of this paper is organized as follows. We start by giving a brief introduction to OSNMA in II focusing on the topics that are needed to understand this paper. Section III explains the rationale behind the design and the SW architecture, and section IV goes more into details regarding the implementation logic. Then section V presents the accompanying command line tools and shows examples of how to use them. Finally, we conclude and summarize the paper in VI.

II. OSNMA OVERVIEW

To achieve the navigation message authentication, Galileo OSNMA adapts the TESLA broadcast authentication scheme. Full details regarding the TESLA protocol can be found from Perrig and Tygar (2003) and the OSNMA specific details can be found from the interface control document (ICD) European Union (2022b).

The goal of the TESLA protocol is to authenticate messages broadcast via an untrusted communication channel. In the OSNMA context, this means authenticating the navigation messages that arrive to the receiver. The messages are authenticated by broadcasting an authentication tag for each message. Then the authentication process involves recomputing the tag and comparing it against the received tag. By design, the authentication tags cannot be verified immediately, but a TESLA key, which is broadcast later, is required to verify a tag. This is why an attacker cannot forge the tags: the tag computation involves the TESLA key, which the attacker does not have access to when the tag is being broadcast. The tags and the TESLA key are broadcast on a subframe basis (every 30s), though the received TESLA key authenticates the tags from the previous subframe. Further more, the TESLA keys form a hash chain, meaning that a received TESLA key can be verified by hashing the key and comparing it against a previously verified TESLA key. Usually the verified TESLA key is the so-called root key, that is regularly broadcast. The root key is verified using a digital signature and to verify this signature, the receiver must know the OSNMA public key, which is available through a web service, or possibly via the SIS Fernández-Hernández et al. (2016). The public key, in turn, is authenticated by a Merkle tree root preinstalled in the receiver Hernández et al. (2019). Fig. 1 illustrates this chain of trust in OSNMA. In conclusion: a variety of well tested cryptographic methods are applied and all of the pieces required for navigation message authentication can be obtained from the SIS, except for the Merkle tree root.

The so-called *cross-authentication* is a notable feature of OSNMA. Galileo satellites transmitting OSNMA data will not only transmit their own tag, but also tags for some other satellites as well. This means that not all satellites need to transmit OSNMA data, and it also adds robustness and redundancy to the system. Multiple satellites may transmit equivalent data, and therefore potential problems in message reception from a single satellite should not affect the overall performance. In the future cross-authentication could be used to authenticate navigation message from other constellations as well.

An important practical note is that different tags may authenticate different parts of the navigation message. Tags are associated

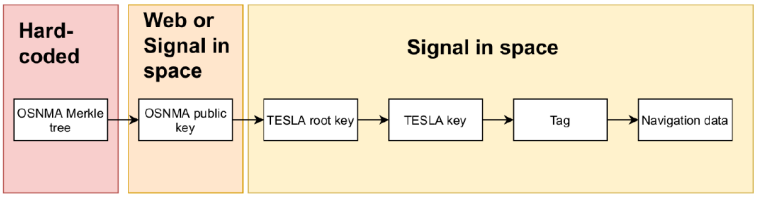

Figure 1: OSNMA chain of trust.

with a so-called ADKD number. The ADKD number specifies what part of the navigation message is authenticated by the tag. Currently the different ADKD values and their meaning are the following.

- ADKD=0: the tag authenticates ephemeris, clock, and status of the satellite.
- ADKD=4: the tag authenticates the Galileo constellation related timing information.
- ADKD=12: the tag authenticates the same information as ADKD=0, but there will be an additional 10 subframe (or 300s) delay in transmitting the TESLA key needed to authenticate the tag.

III. SOFTWARE ARCHITECTURE

In this section we present the SW architecture and rationale of our implementation. In addition to a correct implementation of the authentication logic, the goal is to make a library capable of supporting different receivers, and with a flexible and configurable input and output streams.

1. Rationale

The purpose FGI-OSNMA is OSNMA processing: decoding the OSNMA information from a data stream and authenticating the navigation messages. As such, it will not include a PVT engine, and it cannot be directly used to perform *authenticated positioning*. Of course, authenticated positioning is one of the primary applications of OSNMA, and as such, there are ways to incorporate the authentications produced by FGI-OSNMA into a PVT engine to produce authenticated position. An example of such is shown in subsection V.2.

In other words, the authentication process and the positioning process are decoupled. In order for such system to work and be useful in real-world applications, there must be efficient and seamless ways to pass data from the authentication process to the positioning process.

To achieve this, FGI-OSNMA uses a subscriber-based system, where different events can be handled by a set of subscribers. The subscribers can be used to accomplish many tasks, however, perhaps the most important role of the subscribers is to pass the authentication results to a data sink. This data sink can be a file or a network socket, but also another program like a positioning engine. The subscriber system will be discussed in a more concrete manner in subsection IV.2.

As the focus of the FGI-OSNMA is the OSNMA processing, the main output of FGI-OSNMA are the *authentication events* that it produces. As previously explained, these authentication events can then be broadcast by the subscriber system, and caught by different programs. The different programs can then choose how to use this information. For example, a spoofing detector may only be interested in the failures to authenticate the navigation message. On the other hand, a PVT engine may only be interested in the successful authentications, to know which navigation messages to trust. The authentication events describe the different possible outcomes that can happen during the authentication process. These include the following.

- Success: the authentication of a tag was successful.
- Missing tag: the authentication of a given navigation message cannot be attempted, because we are missing the corresponding tag.
- Missing navigation data: we have received a tag, but not the corresponding navigation data. As such, the authentication cannot be attempted.
- Failure: the necessary data to perform the authentication was received, however, the tag comparison failed. Therefore, the navigation message cannot be trusted.

In addition to the authentication events, FGI-OSNMA can publish other information as well. This can include data transmission events (such as failed Cyclic Redundancy Check (CRC)) or information about the internal state of the engine (such as success or failure to authenticate a TESLA root key).

To reiterate and summarize: the vision is that FGI-OSNMA focuses on the OSNMA processing, and external programs can receive any desired OSNMA related information from the FGI-OSNMA. It is up to the external programs to choose how to use the information. This decoupled nature of FGI-OSNMA reduces the project scope, while still guaranteeing extendability and usability in real-world applications.

2. Architectural Layers

The architectural layers are largely independent of each other, and each layer only requires the output of the previous layer. Fig. 2 shows a schematic representation of the different layers of the SW architecture. The layers are:

- DataSource: Encapsulates the type of input stream used to gather the data. At this stage, the data is simply treated as a raw stream of bytes, without any interpretation/parsing of the content. Classes in this layer implement the methods necessary to retrieve byte streams using different means and make them available to the upper layers following a specified common convention. So far, we have implemented classes that retrieve the input from files, serial ports/USB, and network sockets. The abstract design makes it possible to add other communication methods as well.
- InavPageSource: Encapsulates the steps necessary to parse the data streams provided by the classes of the DataSource layer and extract the nominal I/NAV pages. The classes within this layer then implement the necessary methods for the different receivers that are to be supported. So far only Septentrio receivers are supported. However, adding support for more receivers is simply a question of adding a new class that encapsulates the protocol details of the receiver at hand.
- SubframeSource: In essence, it is a layer with one class that encapsulates the details of how to detect and assemble subframes, from which the OSNMA related information will be retrieved. Note that, thanks to the abstraction and encapsulation provided by the lower layers, this class is receiver independent.
- Authentication: This is the layer that contains the authentication logic. It contains the necessary classes to decode the content of the subframes extracting the necessary information to carry out the navigation message authentication.
- Output: This layer contains classes that convert the output of the authentication in different formats. The purpose is then to decouple the authentication related classes of the particularities of how to pass the resulting information further up in the stack. The delivery method is based on an event-listener paradigm, where upper layer entities interested in the authentication related events are passed the related events in real time.
- DataSink: it receives the formatted outputs from the classes of the Output layer and sends them to the final destination of the information using different communication means. This can mean for example a file, network socket, or one of our tools that uses the authentication results.

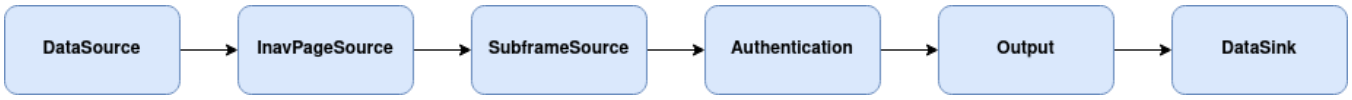

Figure 2: Layers of the software architecture and the flow of data.

The resulting architecture features the necessary abstraction and encapsulation that provides the flexibility necessary to be able to easily reuse the core OSNMA implementation logic in different applications. In particular, it allows a seamless incorporation of different sources of I/NAV messages (such as GNSS receivers or custom files) independently of the communication methods used to retrieve them (USB/serial, files and network readily available), and fully decouples the output of authentication related outcomes from the particularities of how they are to be conveyed to upper layers both, in terms of the communication method (e.g. call an event-listening function, store in a file, send via network, etc) and of the format.

IV. IMPLEMENTATION LOGIC

1. Overview of the Implementation

Each core class of the project has a logical purpose and they are very closely related to specific concepts related to OSNMA protocol. The core classes and their purposes are the following.

- `OsnmaDecoder`: this class handles the extraction and decoding of the OSNMA data from the received navigation pages.
- `OsnmaAuthenticator`: handles the core cryptographic and authentication procedures, such as tag computation or verification of the root key.
- `NavigationDataManager`: collects and manages the navigation data from satellites, which is to be authenticated.

- `OsnmaEngine`: this class utilizes the other core classes to handle the main OSNMA authentication loop. This is similar to what some other related projects call the "OSNMA receiver".
- `SubscriberSystem`: some of the events are handled in a subscriber-publisher pattern. The subscribers will implement some callback functions, which can handle different tasks, for example, passing the authentication results from one program to another.

The classes and their dependencies are further highlighted in Fig. 3.

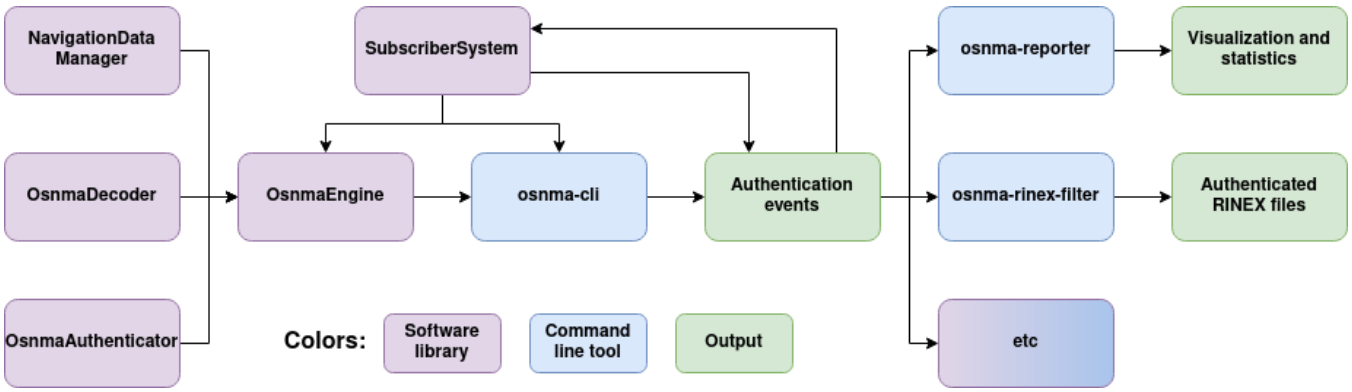

Figure 3: Overview of the core components in FGI-OSNMA.

FGI-OSNMA uses only very basic external dependencies. Most dependencies are general Python modules (such as `os`, `sys`, `socket`) that are included in any Python distribution. The cryptographic functionality comes from modules `cryptography`, `hashlib`, and `hmac`. In addition to this, the bit level operations are done using the `bitstring` module, and the command-line tools use either `argparse` or `configargparse` modules.

2. Publisher-Subscriber System

The goal of the `SubscriberSystem` is to add modularity and handle the communication to external programs, however, most of the internal communication does not go through the `SubscriberSystem`. The `SubscriberSystem` provides a way to execute callback functions upon important events in the authentication process. The callback function can be essentially any Python function, either one included in FGI-OSNMA or one created by the users themselves. While the callback functions need to be written in Python, the external program are not assumed to be written in any particular programming language. However, in case the third-party program is not written in Python, some form of middleware is required to pass the data. Integration of FGI-OSNMA into Python projects can be somewhat simpler compared to projects written in other programming languages, however, there are well-known frameworks to facilitate communication between programs of different programming languages. These popular frameworks include Robot Operation System (ROS) Quigley et al. (2009); Macenski et al. (2022), ZeroC Ice ZeroC (2023), and Redis Redis (2023), but of course, simple network transfer of the raw data and parsing the data on the reception side is also possible. It is also worth noting that the format of the messages that FGI-OSNMA produces is quite simple: the authentication events essentially consist of few numbers and status. Therefore, the needed middleware pieces can be quite simple.

It is possible to register any number of subscribers, and therefore callback functions, to achieve multiple different goals. As an example, one can subscribe to the authentication results, and then use one subscriber to write the results to a file, use another subscriber to pass to authentications to some analysis module, and use a third subscriber to serialize the data and to pass it to a completely different program.

The user can specify what subscribers to use in a configuration file. Due to the configuration file and the modularity of FGI-OSNMA, having a custom subscriber does not interfere with the rest of the system. In particular, the user does not need to touch any of the core source files to create and register custom subscribers. The custom subscribers can be removed easily and it is of course possible to switch between configuration files. A practical example of using a custom subscriber can be found in subsection V.4.

3. Information Flow

In our OSNMA engine, we abstract the information flow from space into four related streams of information, namely:

1. Navigation data stream: This is the stream of navigation data transmitted from space, to be authenticated.

2. Tag stream: The stream of tags authenticating pieces of the navigation data.
3. TESLA key stream: The stream of TESLA keys that are used together with the tags to authenticate the navigation data.
4. Control stream: The stream transmitting the current protocol configuration and a public key signature for a trusted TESLA root key. This stream also handles events such as key and chain revocations.

The information flow is represented in Fig. 4. Here PRNA is the identifier of the satellite transmitting the tag, and PRND is the identifier of the satellite transmitting the navigation message to be authenticated. The ADKD is as defined in section II. The lines show which pieces of data are associated. Associating data from the three streams is required to perform authentication. Although, recall that the keys are associated with the subframes with a 30 second delay.

The tag stream, the TESLA key stream and the control stream are all mixed together in the 40-bit OSNMA data field that gets transmitted within every nominal page from SIS. Each transmitted subframe (15 nominal pages) contains one logical unit of information, so the OSNMA data is most naturally decoded on a subframe-by-subframe basis. However, we found that subframes may often be incomplete (i.e. missing pages). This can happen due to natural reasons, such as signal reception problems due to low satellite elevation, or occlusions in urban canyons. Therefore the information is best processed at a page level, or the receiver should have some other way to cope with incomplete subframes. The advantages of a page level processing were also noted in Damy et al. (2022). On the other hand, signatures in the control stream may span multiple subframes, so coordination across multiple subframes is also needed.

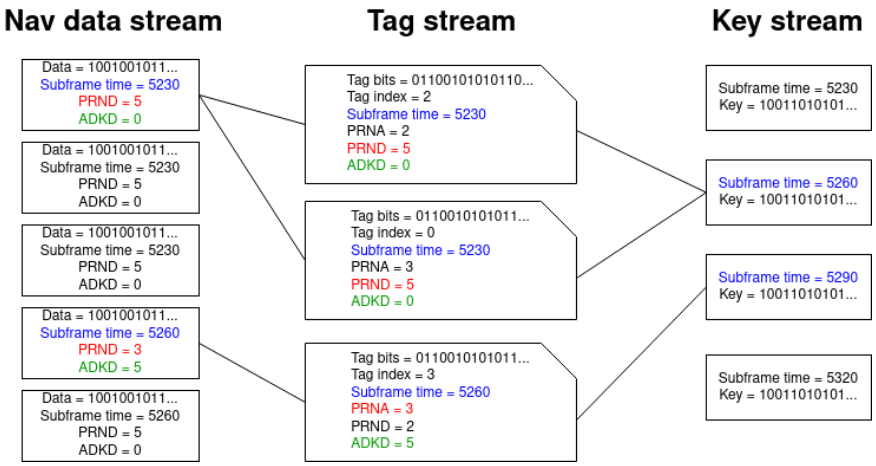

Figure 4: OSNMA Information flow, with the three OSNMA streams from the SIS after the initialization phase. The control stream is not pictured.

Details for decoding streams 2-4 from the 40-bit OSNMA field are given in the latest ICD and receiver guidelines European Union (2022b,a). The decoding is mostly straightforward, but the implementation is complicated by the fact that the key and tag sizes are not fixed, but defined in the control stream, so the keys and tags cannot be parsed before the protocol configuration is decoded from the control stream. Thus, a separate initialization phase is required before authentication can commence. Our implementation stores all the navigation data received during the initialization phase to be able to retroactively decode the data once the protocol parameters have been received.

After initialization, our implementation maintains a buffer containing currently unauthenticated data and their associated authentication tags. When the TESLA key of the next subframe is transmitted, all the data in the buffer is validated and the buffer is emptied, assuming the new key was valid (see Fig. 5). If the key of the next subframe is not valid, or not received at all, we cannot authenticate the data in the buffer. In this case, we keep accumulating the data in the buffer, until a valid TESLA key is received. Even if there is a gap in the chain of keys, we can always iterate the key chain back to a previous trusted key to verify the new key.

In practice, the authentication buffer in our implementation is a hash table, where the keys are triples (t, s, a), where t is a timestamp, s is the identifier (SVID) of the satellite transmitting the data to be authenticated, and a is the ADKD number. The values of the hash table are pairs (A, T), where A is the piece of navigation data specified by the ADKD number, and T is an authentication tag for that data. This way, the tags are efficiently matched to their associated navigation data, ready for authentication when the next TESLA key is broadcast.

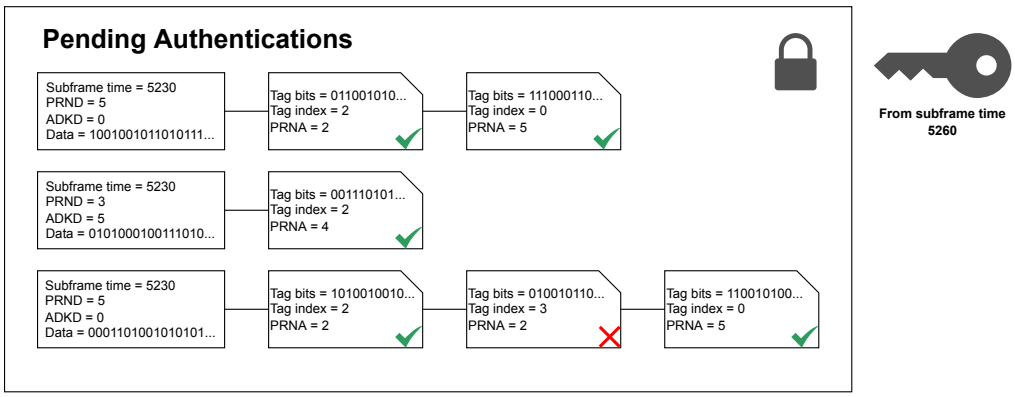

Figure 5: Pending authentication results. Data is collected each subframe and stored in a buffer, where it waits for the new key and authentication. Due to the cross-authentication scheme, the same navigation data may receive multiple corresponding tags to authenticate it. If one tag is invalid (due to natural data corruption, or the data being from a spoofer), the data may still be authenticated by the other tags.

V. USAGE

This section describes the usage of the command line interface, the previously mentioned tools, and the usage is demonstrated in a few real-world use cases. Namely, we describe how to perform authenticated positioning using our tools, and we discuss how FGI-OSNMA was integrated to GNSS-Finland. However, the more in-depth and up-to-date documentation regarding the usage and tools can be found in the FGI-OSNMA software repository. As a Python project the library is cross-platform, but the examples in this section use a Linux environment to demonstrate the command line usage of FGI-OSNMA.

1. Command Line Interface

The command line tool `osnma-cli` is the primary way of using the library. The basic usage can be seen below. However, it is worth noting that none of the arguments are mandatory. If the input and output are not specified, then `stdin` and `stdout` are respectively taken as their values. The root key and public key are needed to start the authentication process, but they are not required as arguments as they can be also retrieved from the SIS. However, the authentication process will start faster if these were provided in the command line arguments. Another important note is that the Merkle tree root is required to authenticate the public keys, which in turn are required to authenticate the TESLA root key, and start the navigation message authentication process. However, if a public key is provided in the command line arguments without a Merkle tree root, then the program will report a warning, but still accept the public key as authentic. This simplifies the usage, and should not pose dangers, as the user should always use either a public key retrieved from European GNSS Service Centre (GSC) or from the SIS.

```
./osnma-cli \           # calling the main program
    -i input \          # specify input
    -p protocol \       # specify the input protocol
    -o output \         # specify output
    -f format \         # specify the output format
    -k public_key \     # specify the path to Galileo public key
    -r root_key \       # specify the path to the current TESLA root key
    -s new_root_key \   # save the root key to the given path upon reception
    -m merkle_tree      # the Merkle tree root
```

Running the above command would start producing the authentication results and reports on other noteworthy events. An example of this is seen in Fig. 6, where multiple instances of the Python class `AuthAttempt` are printed to `stdout`. However, this class could have been as well passed to another program, redirected, or written to a file.

FGI-OSNMA is flexible in terms of its input and output formats, and in addition to this, it works well with Unix pipes and other output redirection methods. The data input and output layers of the architecture were designed to make adding new types of input or output sources simple. There are many arguments that the `osnma-cli` can take, however, most of these arguments have sensible default values. Therefore, typing the command can be much less verbose than the example shows.

```
AuthAttempt(PRND=19, PRNA=19, wn=1209, tow=130830, adkd=0, outcome=<AuthOutcome.OK: 1>)
AuthAttempt(PRND=27, PRNA=19, wn=1209, tow=130830, adkd=0, outcome=<AuthOutcome.OK: 1>)
AuthAttempt(PRND=27, PRNA=21, wn=1209, tow=130830, adkd=0, outcome=<AuthOutcome.OK: 1>)
AuthAttempt(PRND=255, PRNA=19, wn=1209, tow=130830, adkd=4, outcome=<AuthOutcome.OK: 1>)
AuthAttempt(PRND=255, PRNA=4, wn=1209, tow=130830, adkd=4, outcome=<AuthOutcome.OK: 1>)
AuthAttempt(PRND=255, PRNA=21, wn=1209, tow=130830, adkd=4, outcome=<AuthOutcome.OK: 1>)
AuthAttempt(PRND=255, PRNA=11, wn=1209, tow=130830, adkd=4, outcome=<AuthOutcome.OK: 1>)
AuthAttempt(PRND=255, PRNA=10, wn=1209, tow=130830, adkd=4, outcome=<AuthOutcome.OK: 1>)
AuthAttempt(PRND=36, PRNA=19, wn=1209, tow=130830, adkd=0, outcome=<AuthOutcome.OK: 1>)
AuthAttempt(PRND=36, PRNA=4, wn=1209, tow=130830, adkd=0, outcome=<AuthOutcome.OK: 1>)
AuthAttempt(PRND=36, PRNA=11, wn=1209, tow=130830, adkd=0, outcome=<AuthOutcome.OK: 1>)
AuthAttempt(PRND=12, PRNA=19, wn=1209, tow=130830, adkd=0, outcome=<AuthOutcome.OK: 1>)
AuthAttempt(PRND=12, PRNA=4, wn=1209, tow=130830, adkd=0, outcome=<AuthOutcome.OK: 1>)
AuthAttempt(PRND=12, PRNA=10, wn=1209, tow=130830, adkd=0, outcome=<AuthOutcome.OK: 1>)
AuthAttempt(PRND=4, PRNA=4, wn=1209, tow=130830, adkd=0, outcome=<AuthOutcome.OK: 1>)
AuthAttempt(PRND=9, PRNA=None, wn=1209, tow=130830, adkd=0, outcome=<AuthOutcome.NAV_DATA_MISSING: 4>)
```

Figure 6: Authentication event examples from a session.

2. RINEX File Filtering and Authenticated Positioning

A common motivation for authenticating the received GNSS data is to compute an *authenticated* PVT solution. We define an authenticated PVT solution as a PVT solution where the computation has been done by only utilizing authenticated data. In the context of OSNMA, this means that only authenticated navigation messages are used in the computation. As FGI-OSNMA is an OSNMA computation engine, and not a PVT engine, it cannot be directly used to compute authenticated PVT solutions. However, the authentication events produced by FGI-OSNMA can be used in an external PVT engine, and an example of such is presented in this subsection.

The authenticated positioning in this example relies on filtering out unauthenticated navigation messages from a Receiver Independent Exchange Format (RINEX) file. This is accomplished by using a command line tool called `osnma-rinex-filter`, which is included in the FGI-OSNMA software package. This tool can filter RINEX3 files based on the authentication results made by `osnma-cli`. The basic usage of the `osnma-rinex-filter` is highlighted below.

```
./osnma-rinex-filter \      # calling the main program
    -a authentications \    # file containing the authentication results
    -n rinex_nav \          # RINEX navigation file to filter
    -o rinex_obs \          # RINEX observables file to filter
    -p policy               # filtering policy to use
```

This script outputs two files for each given RINEX file: the filtered RINEX file (called `filtered_<original_name>`) and a file containing the dropped messages (called `dropped_<original_name>`).

While OSNMA is meant to authenticate (and therefore filter) only navigation messages, it may also be of interest to remove the observables corresponding to unauthenticated satellites. Therefore, `osnma-rinex-filter` facilitates the filtering of both RINEX navigation and observables files. If the file containing the authentications is not provided, it is taken to be `stdin`. Therefore the RINEX filtering using `osnma-cli` and `osnma-rinex-filter` can be combined into a single step with a Unix pipe.

```
osnma-cli <arguments> | osnma-rinex-filter <arguments>
```

In the next example we demonstrate authenticated positioning by filtering out unauthenticated messages from a RINEX file, and then performing PVT computation with the filtered RINEX file. While real-time performance was a key criterion when developing FGI-OSNMA, this example works in post-processing mode, as the RINEX file based position computation is not real-time by nature.

The RINEX file format is commonly used by the GNSS community for storing the GNSS observables and navigation messages in plain-text format. Most PVT computation engines, such as RTKLIB Takasu and Yasuda (2009), can utilize these RINEX files to compute the PVT solution. The previously mentioned `osnma-rinex-filter` can remove unauthenticated navigation messages from a RINEX navigation file, thus enabling authenticated PVT computation.

The process of using FGI-OSNMA and RTKLIB to perform authenticated positioning involves the following three steps.

1. Run the OSNMA engine (`osnma-cli`) to obtain the authentication events.

2. Run the RINEX filter (`osnma-rinex-filter`) on the target RINEX file. The file containing the authentication events is given as an argument to the program.
3. Run the PVT computation engine (RTKLIB) using the filtered RINEX files.

3. Visualization and Computation of Statistics

The software package contains a tool called `osnma-reporter`, which can be used to visualize authentication events and compute statistics related to an authentication session. In particular, Fig. 7 and the statistics in Table 1 have been created by using the `osnma-reporter` tool.

Fig. 7 shows a visual representation of the authentication information that is produced by FGI-OSNMA, and Table 1 shows the relevant statistics from this authentication session. In particular, Fig. 7 displays the authentication status and type of satellites, other events related to authentication, and finally data transmission related events, such as dropped navigation pages or failed CRCs.

Fig. 7 and Table 1 are based on a approximately one day long dataset collected in open-sky conditions in southern Finland. The data collection was done at the end of August, during which the latest ICD European Union (2022b) was in place. The Septentrio mosaic-X5 receiver with the Septentrio PolaNt Choke Ring antenna were used in the data collection.

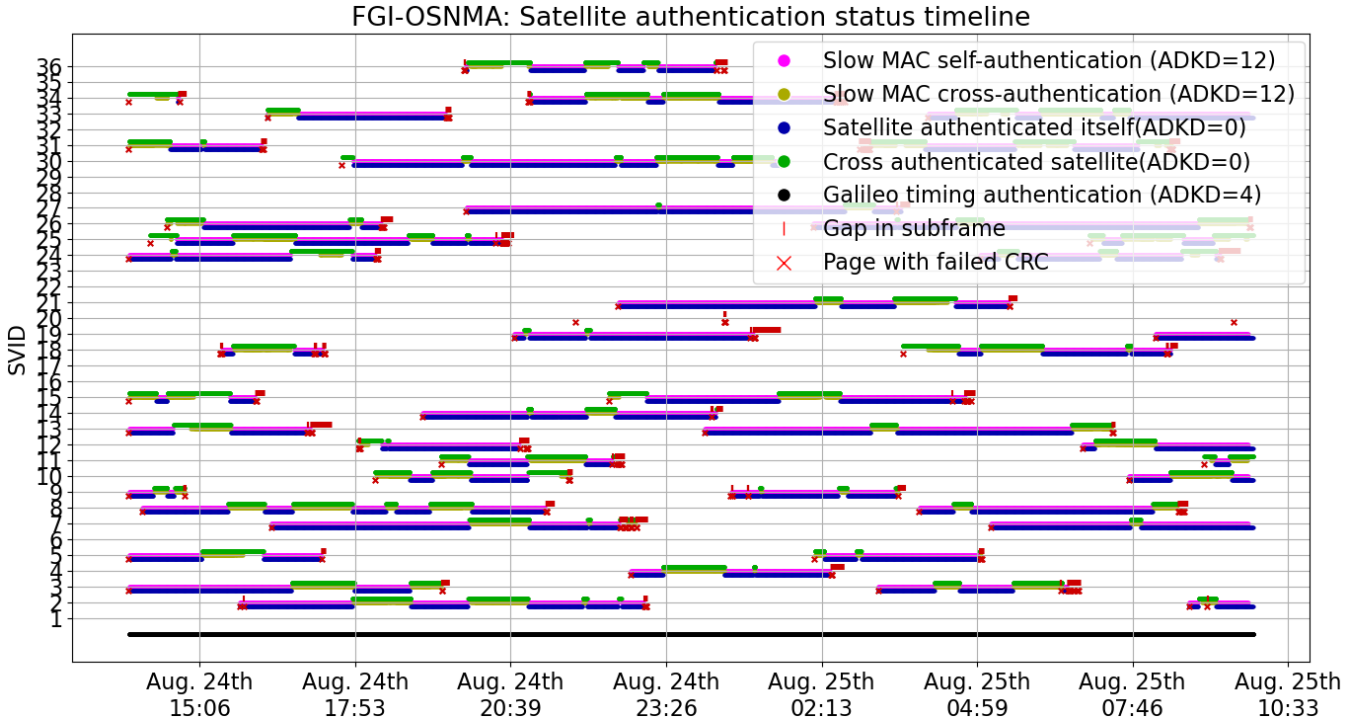

Figure 7: The authentication status timeline produced by the visualization tool.

Table 1: Statistics related to the authentication.

Statistic	Value
Simultaneous authenticated satellites: 5% percentile	7
Simultaneous authenticated satellites: average	9.25
Simultaneous authenticated satellites: 95% percentile	11
Percentage of authenticated fixes	100%

The usage of `osnma-reporter` simple, and if no arguments other than the input file are provided, it will produce all visualizations and statistics. This includes what is seen in Fig. 7 and Table 1, but it can also visualize the number of visible satellites transmitting OSNMA data over time, or the number of simultaneous authenticated visible satellites over time. Support for more visualizations and KPIs may be added in the future, and therefore, the documentation in the software repository should be referred. The usage can be seen below.

```
./osnma-reporter \
    -i authentications \    # file containing authentication events
    --timeline \            # generate a timeline related to authentication events
    --statistics \          # calculate statistics related to the authentications
    --transmitters \        # visualize satellites transmitting OSNMA data over time
    --authentications \     # visualize number simultaneous authentications over time
    --all                   # or -a, use all of the above options (default)
```

4. Usage in GNSS-Finland

To further highlight the integrability of FGI-OSNMA, we demonstrate a real-world use case of FGI-OSNMA: performing OSNMA authentications for the GNSS situational awareness service GNSS-Finland GNSS-Finland (2023).

GNSS-Finland is a website for monitoring the GNSS signal quality in Finland. This involves tracking the signal quality of all of the major constellations and signals, and monitoring the positioning accuracy. GNSS-Finland monitors the data coming from 47 FinnRef stations around Finland. The FinnRef stations are a continuously operating reference station (CORS) network that is operated by the NLS. There is a fully open and free version available at GNSS-Finland (2023), but the following discussion and changes is based on the development server of GNSS-Finland, which is not publicly available. However, we expect to push the changes related to the OSNMA monitoring to the production server later this year.

Getting the authentication events produced by FGI-OSNMA to GNSS-Finland involves writing a custom subscriber. Communication inside GNSS-Finland is based on a subscriber-publisher pattern as well, though this differs from the subscriber-publisher pattern of FGI-OSNMA. In particular, the subscriber system of GNSS-Finland uses the remote procedure call library ZeroC Ice ZeroC (2023). However, the details of the subscriber system of GNSS-Finland are not important for this example, and the main point is that the developers of GNSS-Finland just need to make a simple middleware between GNSS-Finland and FGI-OSNMA.

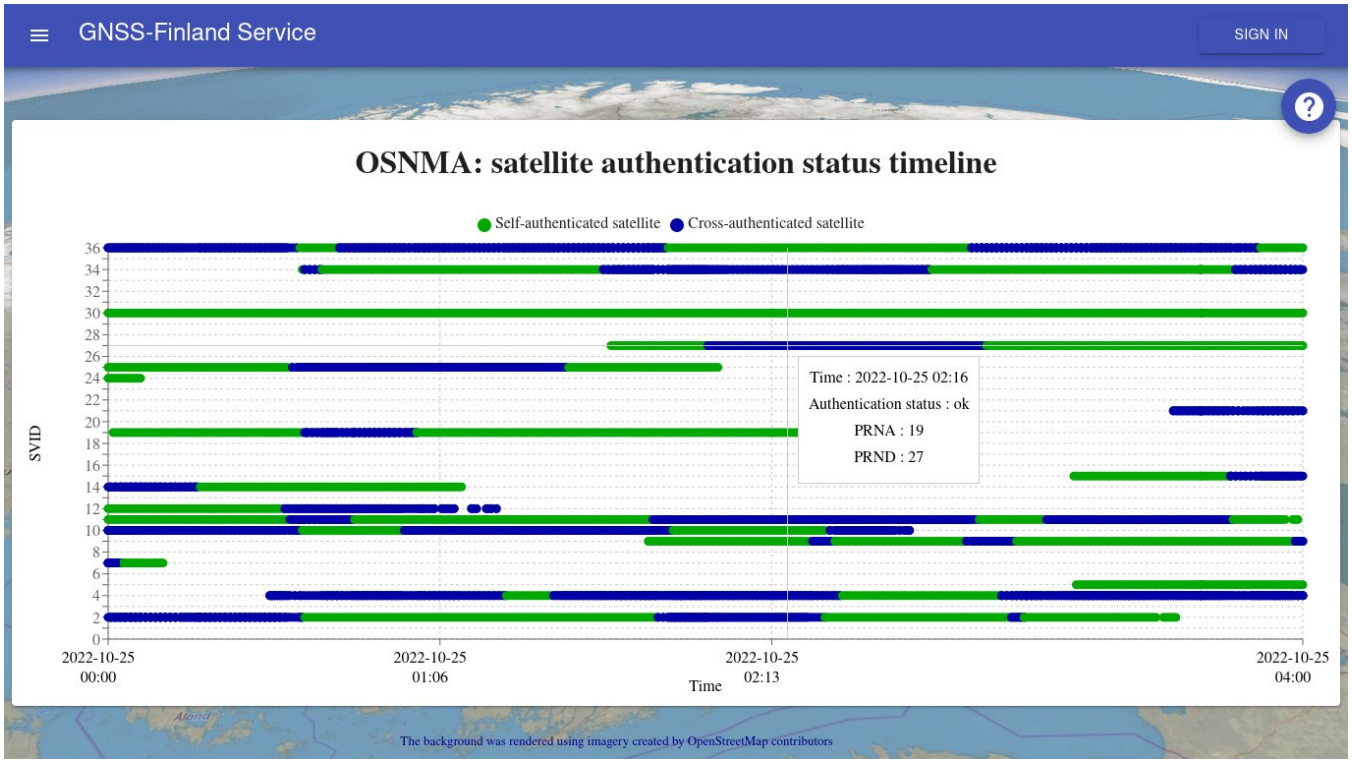

Figure 8: OSNMA authentication timeline in GNSS-Finland

The complexity and the format of the middleware will naturally depend on the target software project, but in the case of GNSS-Finland, the authentication events are passed to an application programming interface (API) call. The API call will then store the events in the database of GNSS-Finland, from which it will be accessible to the rest of the system. Of course, preliminary steps of preparing a database table and creating a front-end module to visualize the results were needed. After the preliminary steps and the creation of the custom subscriber are done, simply executing the `osnma-cli` will start feeding the

authentication events to the API and database of GNSS-Finland. As mentioned before, the active subscribers are designated in a configuration file, and it is easy to remove the GNSS-Finland specific subscriber if desired.

Fig. 8 shows an early prototype of the authentication status monitoring of GNSS-Finland utilizing FGI-OSNMA.

VI. CONCLUSIONS

We presented an open source Galileo OSNMA implementation called FGI-OSNMA. Special emphasis was put into the modularity and the integrability of FGI-OSNMA. The integrability is enabled by the subscriber-publisher paradigm used in the project. FGI-OSNMA correctly implements the OSNMA specification and tries to be extendable and usable in real-world applications. The real-world usage was highlighted by presenting practical examples, such as performing authenticated positioning and integrating FGI-OSNMA into an external project called GNSS-Finland. Overall, the FGI-OSNMA is a self-contained and flexible software project, and we believe that it can be a helpful tool for the GNSS community.

ACKNOWLEDGEMENTS

This work was conducted in the scope of the ESRIUM Project, which has received funding from the EUSPA as part of EU-Horizon 2020 research and innovation programme under grant agreement No 101004181.

REFERENCES

Bhuiyan, M. Z. H., Söderholm, S., Ferrara, G., Kirkko-Jaakkola, M., Kuusniemi, H., Lopez-Salcedo, J. A., and Fernández-Hernández, I. (2022). *Galileo E1 Receiver Processing*, page 140–152. Cambridge University Press.

Borre, K., Fernandez-Hernandez, I., Lopez-Salcedo, J. A., and Bhuiyan, M. Z. H. (2022). *GNSS Software Receivers*. Cambridge University Press, Cambridge.

Damy, S., Cucchi, L., and Paonni, M. (2022). Performance assessment of galileo osnma data retrieval strategies.

ESRIUM (2023). Esrium project web page. `https://esrium.eu/`. Accessed: 2023-01-31.

Estevez, D. (2022). galileo-osnma. `https://github.com/daniestevez/galileo-osnma`.

European Union (2022a). Galileo open service navigation message authentication OSNMA receiver guidelines.

European Union (2022b). Galileo open service navigation message authentication (OSNMA) signal-in-space interface control document (SIS ICD).

Fernández-Hernández, I., Rijmen, V., Seco-Granados, G., Simon, J., Rodríguez, I., and Calle, J. D. (2016). A navigation message authentication proposal for the galileo open service. *NAVIGATION, Journal of the Institute of Navigation*, 63(1):85–102.

FGI (2023). Repository of the fgi-osnma. `https://github.com/nlsfi/fgi-osnma`. Accessed: 2023-08-31.

Galan, A., Fernandez-Hernandez, I., Cucchi, L., and Seco-Granados, G. (2022). OSNMAlib: An open python library for galileo osnma. In *2022 10th Workshop on Satellite Navigation Technology (NAVITEC)*, pages 1–12.

GNSS-Finland (2023). Gnss-finland web page. `https://gnss-finland.nls.fi`. Accessed: 2023-01-31.

Hammarberg, T., García, J. M. V., Alanko, J. N., and Bhuiyan, M. Z. H. (2023). An experimental performance assessment of galileo osnma. In *2023 International Conference on Localization and GNSS (ICL-GNSS)*, pages 1–7. IEEE.

Hernández, I. F., Ashur, T., Rijmen, V., Sarto, C., Cancela, S., and Calle, D. (2019). Toward an operational navigation message authentication service: Proposal and justification of additional osnma protocol features. In *2019 European Navigation Conference (ENC)*, pages 1–6. IEEE.

Horst, O., Kirkko-Jaakkola, M., Malkamäki, T., Kaasalainen, S., Fernández-Hernández, I., Moreno, A. C., and Díaz, S. C. (2022). Haslib: An open-source decoder for the galileo high accuracy service. In *Proceedings of the 35th International Technical Meeting of the Satellite Division of The Institute of Navigation (ION GNSS+ 2022)*, pages 2625–2633.

Macenski, S., Foote, T., Gerkey, B., Lalancette, C., and Woodall, W. (2022). Robot operating system 2: Design, architecture, and uses in the wild. *Science Robotics*, 7(66):eabm6074.

Motallebighomi, M., Sathaye, H., Singh, M., and Ranganathan, A. (2022). Cryptography is not enough: Relay attacks on authenticated gnss signals. *arXiv preprint arXiv:2204.11641*.

Perrig, A. and Tygar, J. (2003). Tesla broadcast authentication. In *Secure Broadcast Communication*, pages 29–53. Springer.

Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Wheeler, R., Ng, A. Y., et al. (2009). Ros: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, page 5. Kobe, Japan.

Redis (2023). Redis website. `https://redis.io/`. Accessed: 2023-06-29.

Takasu, T. and Yasuda, A. (2009). Development of the low-cost rtk-gps receiver with an open source program package rtklib. In *International symposium on GPS/GNSS*, volume 1, pages 1–6. International Convention Center Jeju Korea Seogwipo-si, Korea.

ZeroC (2023). Product page of zeroc ice library. `https://zeroc.com/products/ice`. Accessed: 2023-06-29.