Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Informatik
Computer Science

Authenticated Positioning Using GALILEO Open Service Navigation Message Authentication (OSNMA)

Semester Project

Lyubomir Kyorovski

September 30, 2022

Advisors:
Martin Kotuliak
Simon Erni
Prof. Dr. Srdjan Čapkun

Department of Computer Science, ETH Zürich

Abstract

Problems with the security of GNSS have been made apparent by academics and bad actors alike, with one of the most serious attacks against it being spoofing of navigational data, which trick an unsuspecting user of such a system into receiving incorrect information about their location. In this paper, we explore one of the mechanisms introduced to provide data authentication to GNSS - Galileo OSNMA - such that the data received by the user can be proven to be originating from a satellite rather than from a malicious party. The document presents an example implementation of the protocol and outlines possible flaws in its design that could undermine the security properties proposed for Galileo OSNMA. We make suggestions on how to make it easier for developers to implement and test OSNMA processing in receivers. We also introduce the concept of an official (reference) implementation of an OSNMA processing library.

Acknowledgments

I would like to thank my supervisors, Martin Kotuliak and Simon Erni, and by extension the whole System Security Group at ETH Zurich, for the guidance, advice and assistance they have provided during the course of this project.

Thank you to all friends and family who helped me keep things in perspective.

Contents

Chapter 1

Introduction

Global Navigation Satellite Systems (GNSS) have been introduced in the last century and since then they have been widely adopted, for both commercial and private use. Applications vary from geo-tagging images and videos to futuristic concepts like autonomous driving, which require not only accurate positioning, but also for it to be secure and reliable. The security of GNSS has been shown to be problematic, with one of the most serious issues being spoofing attacks [1]. Although protected signals exist, such as the Global Positioning System (GPS) M-Code[1], their use is limited to government and military applications. With some consumer products becoming more reliant on GNSS positioning, a publicly available mechanism for improving the defense against GNSS attacks is necessary.

Galileo, an European GNSS, similar to the well-known GPS, provides a new key feature - Open Service Navigation Message Authentication (OSNMA), providing the ability for a receiver to authenticate the messages coming from the satellites using cryptographic primitives [2]. This makes it not possible to generate false navigation messages by an attacker and easily spoof the location of the receiver. Galileo satellites have recently started to broadcast OSNMA data, since the feature entered its public observation (test) phase, so now is a pivotal time for research into its effectiveness and security to take place. Sadly, there is a shortage of OSNMA receiver implementations which researchers need to inspect and manipulate OSNMA data. This paper aims to help fill this void.

This paper is outlined as follows: First, we provide background on GNSS and a brief summary of Galileo OSNMA. Then, we implement the complete navigational data verification process and provide experimental results. We discuss the possible ways in which GNSS receiver developers can be aided in the implementation process. We also suggest possible pitfalls in the design of OSNMA, which could still allow for spoofing attacks. As a final contribution, we introduce the concept of a reference OSNMA library, and modify a software for GNSS signal processing to incorporate OSNMA processing, as a proof-of-concept.

[1] https://www.everythingrf.com/community/what-is-m-code

Chapter 2

Background

The following sections introduce some general knowledge about GNSS, Galileo GNSS and a brief description of OSNMA.

2.1 GNSS

There are multiple operational GNSS constellations(GPS, Galileo, GLONASS, BeiDou, etc.), but all of them share some fundamental concepts and some pitfalls. As this is not a book on GNSS, only the ones relevant to the matter at hand are presented. For a detailed look at GNSS, the reader is referred to [3], which was used as the source material.

2.1.1 Principle of work

GNSS signals are carried by radio waves in the spectrum between 1.2 and 1.6 GHz. These signals are transmitted by medium-earth orbit satellites. More than one satellite (even one from a different GNSS) can transmit on one frequency, through the use of code-division multiple access (CDMA) - each satellite has a unique Pseudo-random Noise (code) (PRN) code, which a receiver can use to determine which satellites it receives signal from.

The PRN code is repeated at predefined intervals, of a few milliseconds to seconds. Provided the receiver knows exactly the time at which the satellite began transmitting this code, subtracting it from the time it began receiving it yields the time necessary for the radio wave to travel from the satellite to the receiver. As electromagnetic waves travel with (roughly) the speed of light, we can also approximate the distance between the two.

The satellites transmit fairly little navigational data (usually only about 50 bits/second), which describes the orbit of a single or multiple satellites. This allows the receiver to know exactly(or at least approximately) where a satellite is situated in space at a given moment in time relative to Earth's surface. There are two types of data that can describe a satellite's orbit - an *ephemeris* and an *almanac*. An ephemeris describes the orbit of a single satellite and is updated regularly, having a short time of validity. An almanac, on the other hand, describes the orbits of all satellites(think of it as a collection of ephemerides) - it is less precise and has longer time of validity. A receiver can store it, so that it can quickly approximate a position even after being turned off for a long time - there will be no need to wait for the ephemeris to be received. Satellites typically transmit their own ephemeris, as well as the almanac of the GNSS. Part of the navigational data are also time and clock correction parameters, so that the receiver can correct the measurement of the distance between it and the satellite.

Since the receiver is able to know where satellites are and how far away they are from it, it can perform trilateration to find out its position.

2.1.2 Attacks on GNSS and countermeasures

GNSS satellites, are relatively far from earth (20000 - 25000 km), so the signal that reaches a receiver is fairly weak, often complemented with a lot of noise. A bad actor, having the right equipment, can easily "drown out" the original signal and perform a variety of attacks.

The simplest attack is injecting noise into the carrier signal, a technique known as *jamming* [4], which is a form of denial of service, as the receiver can no longer filter out the signal from the noise and process it.

The more dangerous types of attacks are spoofing and relay attacks [5, 1, 4], as these can alter the state of a GNSS receiver, fooling the system/person using it and/or being part of a multi-step attack [6, 7]. In relay attacks, the attacker records a signal at location A and quickly retransmits it at location B. The receivers at location B receive the signal and are fooled into computing that they are at location A. Spoofing attacks are even more powerful - by carefully crafting a signal, an attacker can trick a receiver into computing a location of their choosing. This is inherently more complex and can mean modifying the navigational data, precisely timing the transmission of PRN codes, etc.

Although not universally adopted, methods for non-cryptographic protection against these attacks are proposed in [3, 5, 4, 1, 8]. An example strategy is monitoring signal characteristics, e.g. whether the signal is not too powerful to be coming from space, is it coming from an expected angle, has the computed position/velocity rapidly changed, etc. Notably, these mechanisms do not require a modification to the GNSS, but only to the software/hardware of the receiver.

2.2 Galileo GNSS

Galileo is a GNSS operated by the European Union Agency for the Space Programme (EUSPA). The system went live in December 2016[1] and has been operational since.

2.3 System parameters

The system is planned to consist of 30 satellites (24 active + 6 spares). The state of the system as of September 2022 is: 28 satellites in orbit, out of which 23 are usable, 1 is not available, and 4 are unusable[2]. These satellites orbit at 23222 km, the highest of any GNSS.

The reference time in Galileo GNSS is called Galileo System Time (GST), and its start epoch is defined as 13 seconds before midnight between 21st August and 22nd August 1999, i.e. GST was equal to 13 seconds at 22nd August 1999 00:00:00 UTC [9].

According to Galileo's Signal-in-Space Interface Control Document [9], the navigation signals are transmitted in four bands - E1(carrier 1575.420 MHz), E6(carrier 1278.750 MHz), E5a(carrier 1176.450 MHz), E5b(carrier 1207.140 MHz), with E5a and E5b being combined into the high-bandwidth E5 band. Galileo E1 and GPS L1 are on the same carrier frequency, but Galileo E1 has a higher bandwidth.

Galileo Transmits 3 types of messages:

1. **C/NAV** - transmitted on the E6 band, designated for commercial use
2. **F/NAV** - transmitted on the E5a band, part of the open service
3. **I/NAV** - transmitted on the E5b and E1 bands, for both the open and commercial service

In this paper, we will look more closely at the I/NAV message in the E1 band.

[1] https://defence-industry-space.ec.europa.eu/eu-space-policy/galileo_en
[2] https://www.gsc-europa.eu/system-service-status/constellation-information

2.4 E1 I/NAV message structure

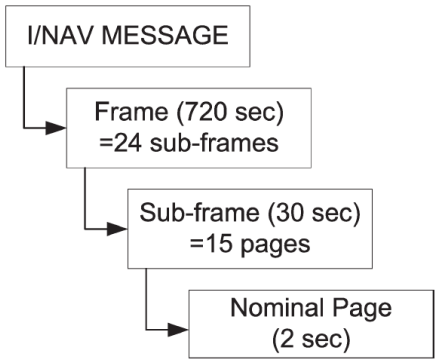

Figure 2.1: I/NAV Message Structure in the Nominal Mode, [9]

The structure of the message, along with the transmission duration of each of its components, can be seen in Figure 2.1. Each second, an even or odd page is transmitted (120 bits). When an even page is followed by an odd page, a nominal page is formed (2 seconds, 240 bits). 15 nominal pages form a sub-frame (30 seconds, 3600 bits), and 24 sub-frames form a frame (720 seconds, 86 400 bits).

Even/odd=1	Page Type	Data j (2/2)	Reserved 1	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

Figure 2.2: I/NAV Nominal page with bits allocation (even page at the bottom, odd at the top) [9]

Figure 2.2 presents the bit allocation in the I/NAV pages. Inside the even pages, there is an 112-bit long data field, and inside the odd there is another 16-bit one. Concatenated together, they form a 128-bit nominal word, which carries navigational data. There are several word types:

- Word Type 0 - I/NAV Spare Word, current GST
- Word Types 1-4 - carry the satellite's ephemeris, clock correction parameters and Space Vehicle ID
- Word Type 5 - Information about signal quality, current GST, other correction parameters
- Word Type 6 - GST-UTC conversion parameters

- Word Types 7-10 - Galileo Almanac
- Word Type 16 - Reduced Clock and Ephemeris Data parameters
- Word Types 17-20 - Forward-error-correction Reed-Solomon for Clock and Ephemeris Data

The words in a sub-frame, and therefore the pages, follow a predefined sequence. We can use this to track when a sub-frame begins and ends. The sequence can be found in Table 2.1.

Time(s)	1	3	5	7	9	11	13	15	17	19	21	23	25	27	29
Word Type	2	4	6	7/9	8/10	0/17/18	0/19/20	0/16	0	0	1	3	5	0	0/16

Table 2.1: Galileo I/NAV message sequence

The 24-bit cyclic redundancy check (CRC) code, which can be seen in Figure 2.2, is computed on all fields of the nominal page, except for the SSP and Tail fields.

2.5 Galileo Open Service Navigation Message Authentication (OSNMA)

Galileo's Open Service, since November 2021, is testing a mechanism for authentication of navigational messages, which allows users to confirm that the data received by their devices originated from Galileo GNSS, and has not been spoofed or modified by a third party. The following is a brief summary of the user ICD and receiver guidelines of Galileo OSNMA [10], [11].

2.5.1 Timed Efficient Stream Loss-Tolerant Authentication (TESLA)

At the core of OSNMA lies the Timed Efficient Stream Loss-tolerant Authentication (TESLA), which was originally designed to provide authentication for multimedia broadcast applications [12]. GNSS conforms to this type of applications, because of its multicast (one-to-many) nature. TESLA employs a combination of symmetric and asymmetric cryptography to sign the data that needs to be authenticated.

TESLA is computationally cheap and can authenticate a lot of data with minimum up-stream transmission, by disclosing the keys with some predefined delay. A plaintext message, in this case navigational data, is authenticated using a Message Authentication Code (MAC). A MAC is associated with a key K_i, which is used to generate it. First, the MACs are disclosed (published), and only after that the keys themselves are made available. These keys are generated using a key-chain, which is created by applying a cryptographically strong hashing function F, starting from a seed key K_n. This chain is represented in Figure 2.3.

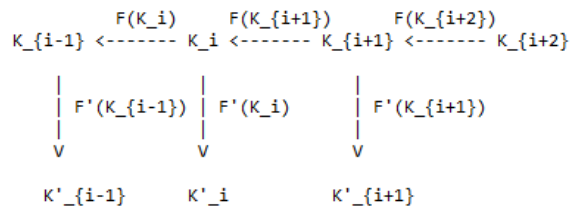

Figure 2.3: TESLA Key chain sequence [12]

The keys of the TESLA chain are transmitted exclusively after the plaintext and its associated MAC, generated using the key, are published. The chain is disclosed in reverse order, starting from K_0 and ending with the seed key K_n. In doing so, messages cannot be verified before

the associated key is received and malicious MACs cannot be generated. Earlier keys in the chain cannot be derived from ones already received, due to the one-way nature of the hashing function F.

The root key, K_0, which is the first one disclosed, when signed, allows for implicit authentication of all subsequent keys, since one could check whether a key at time K_i, by applying F i times, can be used to derive K_0. Therefore, the signature of K_0 must be verified only once, which can be done using the computationally more expensive asymmetric cryptographic primitives. In asymmetric cryptography, signatures are produced using a private key, known only to the sender, and are verified with a public key, known to all recipients. Through signature verification, recipients will be able to trust a message, as only the genuine message issuer, possessing the private key, is able to produce an authentic signature.

2.5.2 OSNMA message structure

The OSNMA message is transmitted through the Reserved 1 field in the E1 I/NAV message page (Figure 2.2). A complete OSNMA message can be defined as all the Reserved 1 fields in a sub-frame, for a total of $15 * 40 = 600$ bits. The Reserved 1 field will be referred to as an OSNMA page.

There are 2 types of messages in an OSNMA message - the HKROOT message and the MACK message. In an OSNMA page, the first 8 bits are part of the HKROOT message and the following 32 bits are part of the MACK message. Concatenating them results in a $15 * 8 = 120$ bit HKROOT message and a $15 * 32 = 480$ bit MACK message. As the fields in these messages can be spread across multiple OSNMA pages, it is important that they are received in order. We can keep that order by tracking the word type sequence from the I/NAV nominal page, as presented in Table 2.1.

HKROOT message

The HKROOT message is 120 bits long and consists of an NMA header, DSM header, and a DSM block.

The NMA header announces the service status of OSNMA, the ID of the TESLA key chain in force, and the status of the key chain and public key in force.

The DSM header identifies the DSM block. It can be marked as either a DSM-KROOT or a DSM-PKR block. Multiple DSM blocks form a DSM message, but they can be transmitted out-of-order, so the DSM Header also tells exactly at which position the current DSM block is in the DSM message. Multiple satellites can transmit different blocks of the same DSM message.

The DSM blocks, once they form a DSM message, depending on the DSM header, can be interpreted as either a DSM-KROOT or DSM-PKR message.

DSM-KROOT messages are responsible for changing the root key of the TESLA chain - this happens when a chain comes to an end, or when the root key is moved to a closer time, so that the receiver can maintain shorter TESLA chains and thus perform fewer cryptographic operations, reducing computational cost(a concept introduced as "floating KROOT" - this should occur at least once per day and at most once per hour). The DSM-KROOT message contains chain parameters, such as the chain ID, Hash function, MAC function, Key size, MAC tag size, time of applicability of KROOT, the KROOT(TESLA key) itself and the digital signature over the KROOT, along with the ID of the public key associated to the signature. Further details on the contents and format can be found in [10].

DSM-PKR messages are responsible for changes to the public key used to verify the KROOT. The authenticity of the transmitted new public key also needs to be verified, for which OSNMA employs a Merkle tree, the root of which is retrieved from the Galileo Service Centre. The data transmitted in the DSM-PKR message is the new public key, its node ID in the Merkle tree, and the intermediate nodes in the tree. The authors of OSNMA note that this type of message will almost never be transmitted.

MACK message

The MACK message is 480 bits long, transmitted every sub-frame. It contains several truncated MAC tags, with associated specific information data(Tag-Info) and a TESLA key. The Tag-Info dictates how the tag must be generated - it can specify exactly which navigational data is being authenticated, as well as from which satellite the OSNMA processor should take that data - which enables cross-authentication.

2.5.3 OSNMA data processing

DSM-PKR Verification

The user is required to authenticate an ECDSA public key or service message received in the DSM-PKR, against a Merkle root, using the hashing algorithm that was used for the tree generation (currently SHA-256). The system might use SHA3-256 in the future for tree generation. The Merkle root can be loaded into the receiver in the factory or retrieved from the OSNMA server with the associated information of which algorithm is to be used for the tree generation.

A leaf of a Merkle tree m_i, identified by message ID, is generated as follows:

$m_i = NPKT||NPKID||NPK$

- $NPKT$ - New Public key type
- $NPKID$ - New Public key ID
- NPK - New Public key

OSNMA uses a Merkle tree that authenticates 16 leaves. A leaf of the Merkle tree m_i is validated against the Merkle root $x_{4,0}$ using the intermediate nodes of the tree $x_{i,j}$.

The base nodes of the tree $x_{0,i}$ are computed as follows:

$x_{0,i} = SHA\{2,3\} - 256(m_i)$

The other nodes $x_{i,j}$ are computed as:

$x_{i,j} = SHA\{2,3\} - 256(x_{j-1,2i}||x_{j-1,2i+1})$

- $j = 1 \ldots 4$ - level in the tree
- $i = 0 \ldots 2^{(4-j)} - 1$ - for each j

To verify the Merkle tree leaf m_i, the nodes have to be computed until reaching the root $x_{4,0}$, then the obtained value should be compared with the stored one.

DSM-KROOT Verification

The user is required to verify the signature of the root key of the TESLA chain, so that it can be implicitly verified that all other keys in the chain are authentic (when they arrive).

Compute the message:

$M = NMA_Header||CIDKR||...||KROOT$

Compute P - zero padding so that the input for the signature algorithm fits an integer number of bytes.

$P = set\_zero(signature\_length\_bits(PK_{OSNMA}) - length(M))$

Then verify with ECDSA: $ecdsa\_verify(M||P, PK_{OSNMA})$.

TESLA Key Verification

TESLA keys belong to a chain that starts with a random seed key K_n, which is only known by the OSNMA provider, and ends with a root key K_0 that is public and certified through the DSM-KROOT.

F is a function that when applied n times to a key K_n yields key K_0, i.e., each element of the chain can be constructed by applying F to the previous element:

$K_0 = F(K_n, n)$ $K_i = trunc(l_k, HF(K_{i+1}||GST_{SF_i}||\alpha))$

- i - index of the key in the chain
- l_k - key length in bit
- α - the 48-bit unpredictable value, consult [10]
- GST_{SF_I} - Galileo System Time at the start of the 30-second sub-frame in which the key K_i is transmitted. For Galileo E1 I/NAV, this is the start time of the E1 sub-frame minus 1 second. When evaluating K_0, the formula is instead $GST_{SF_I} = GST_0 - 30$, where GST_0 is the time of applicability associated with the chain.

A TESLA key K_i can be verified against a root key K_0, by computing $F(K_i, i)$ and comparing the result with K_0. The number of hashes to be performed in this case is

$i = \lfloor \frac{GST_{SF_i} - GST_0}{30} \rfloor + 1$

MAC Look-up Table Verification

The tag ADKD sequence can be partially or totally fixed for each chain through the MAC look-up table ([10], section 3.2.3.8). When fixed, the ADKD type of the tag being verified shall match the one indicated in the look-up table. The value to be compared is generated by appending the ADKD type, e.g. 04 or 12 to a string signifying whether cross- or self-authentication is done - E or S, respectively. This is done for each Tag-Info in the MACK message. The string sequence is then compared to the relative sequence from the table in [10], Annex C.

MACSEQ Verification

The 12-bit MACSEQ authenticates the Tag-Info fields of those tags received in the MACK message and whose ADKD type is flexible as per the tag sequence. The MACSEQ is generated with the same key and MAC function as the rest of the MACs in the same MACK message and is verified by comparing the received value with a value computed locally as follows:

$MACSEQ = trunc(12, mac(K, m))$

- K - Key from the TESLA chain used for the Tag_0 generation
- $m = PRN_A||GST_{SF}$ - In case there are no tags defined as flexible in the MACLT entry
- $m = PRN_A||GST_{SF}||MFLEX_1||...||MFLEX_n$ where:

- PRN_A - 8-bit unsigned integer that identifies the satellite transmitting the authentication information and it takes always the value of the SV_{ID} of the Galileo satellite transmitting the information.
- GST_{SF} - the start of the 30-second sub-frame in which the MACSEQ field is transmitted and is represented as a 32-bit unsigned integer.
- $MFLEX_{1...n}$ - Tag-Info fields to be authenticated. $MFLEX_i$ represents the Tag-Info field of the i-th tag in the current MACK message defined as flexible within the sequence provided by the MAC Look-up Table.

Tag Verification

All tags, except Tag_0 are generated as follows:

$tag = trunc(l_t, mac(K, m))$

- l_t - length of the tag
- K - key from the TESLA chain. This key is normally found in the sub-frame received after the one with the MAC tag. In case of slow MACs (ADKD type 12), the key is taken with a delay of 10 sub-frames, meaning the 11th after the one with the MAC tag.
- $m = PRN_D||PRN_A||GST_{SF}||CTR||NMAS||navdata||P$; unique for each tag
  - PRN_D - the PRN of the satellite transmitting the navigation data to be authenticated and is provided with the corresponding Tag-Info section. In the case $PRN_D = 255$, it is set to $PRN_D = PRN_A$
  - PRN_A - the PRN of the satellite transmitting the authentication data
  - GST_{SF} - the start of the 30-second sub-frame in which the tag is transmitted and is represented as a 32-bit unsigned integer
  - CTR - 8-bit unsigned integer identifying the position of the tag within the MACK message - it has value of '1' for the first tag in the tag sequence, incremented by 1 for each subsequent tag
  - $NMAS$ - the NMA status message transmitted within the NMA header
  - $navdata$ - concatenation of the navigation data from the previous sub-frame being authenticated, for the corresponding ADKD indicated within the Tag-Info field. OSNMA is able to authenticate the ephemeris, the clock correction parameters, and timing parameters. For the specific fields being authenticated, see [10], Annex B.
  - P - zero padding sequence such that the length of m fits the minimum number of integer bytes

In case of Tag_0:

$Tag_0 = trunc(l_t, mac(K, m_0))$

$m_0 = PRN_A||GST_{SF}||CTR||NMAS||navdata||P$, with $CTR = 0b00000001$

2.5.4 OSNMA requirements

To comply with the TESLA protocol and guarantee the authenticity of the data, the receiver must ensure that the navigational data message and its MAC tag are received before the OSNMA system discloses the TESLA chain key. This implies that the receiver has to be synchronized to GST, with some tolerance, before receiving and processing OSNMA data. The

time synchronization requirement T_L for OSNMA is set to 30 sec. If the receiver verifies this condition, all authentication types and all tags can be used for the authentication of navigation data.

If this condition is not verified, slow MACs, i.e. messages whose associated TESLA chain key is transmitted with an extra delay, may be put to use. A receiver synchronized with GST with an accuracy better than $T_L + 300$ can process slow MAC with a 10 sub-frame delay. If the receiver doesn't fulfill any of the time synchronization requirements listed above, the OSNMA protocol must not be used.

Receivers should be able to download public key information, most importantly Merkle roots, during the OSNMA lifetime, in case the Merkle tree is renewed. Therefore, the OSMA guidelines [11] highly recommend that the receiver has an internet connection. An added benefit to having an internet connection is the fact that receivers can use clock synchronization protocols such as the Network Time Protocol (NTP) [13] to consistently fulfilll the time synchronization requirements of OSNMA

The receiver should also be able to perform the following cryptographic algorithms:

- Elliptic Curve Digital Signature Algorithm (ECDSA) - P-256 and P-521
- Secure Hash Algorithm 2 (SHA-2) - SHA-256, SHA-512
- Secure Hash Algorithm 3 (SHA-3) - SHA3-256
- Hash-based Message Authentication Code (HMAC) - HMAC-SHA-256
- Cipher-based Message Authentication Code (CMAC) - CMAC-AES

With regard to the sender, i.e. Galileo GNSS, it is required that the Galileo ground network communicates with OSNMA-enabled satellites. However, the ground network does not have the capacity to be connected to all satellites, so currently not all Galileo satellites can have OSNMA enabled.

Chapter 3

Implementation

3.1 Software for capturing OSNMA samples

In order to perform OSNMA processing, naturally OSNMA data has to be retrieved first. To obtain samples with the I/NAV nominal pages and OSNMA data, two methods were developed, described in the following subsections. They are stored as a CSV file, each row having the following structure:

```
osnma,[PRN],[Word Type],[HKROOT Message],[MACK message],[navdata]
```

These are interpreted as follows:

- **osnma** - identifies osnma CSV data, for the purpose of filtering out unnecessary output
- **PRN** - integer, identifying the satellite from which the data in the row originates
- **Word Type** - integer, identifying the type of navigational data in the row, also used to track sub-frame boundaries.
- **HKROOT Message** - string of 8 bits, containing the HKROOT message
- **MACK message** - string of 32 bits, containing the MACK message
- **navdata** - string of 128 bits, containing the navigational data word

3.1.1 Modified version of GNSS-SDR

GNSS-SDR [14] is an open-source software-defined GNSS receiver based on GNU Radio[1] and written in C++, capable of receiving navigational messages from all popular GNSS constellations, and is able to compute positional data from them. It supports various devices for the RF-front-end, such as RTL-SDR, HackRF One and the USRP device family.

Galileo I/NAV message parsing is implemented in `src/core/system_parameters/galileo_inav_message.cc` in the function `split_page`. It was modified to print the data necessary for OSNMA processing in the CSV format described above, logging it into the standard error output to make it easier to filter it from the standard GNSS-SDR output. The data is logged only after the CRC check has passed, as per the OSNMA guidelines.

Due to a bug in GNSS-SDR, the first few pages received for a satellite always have their PRN set to 0. As the correct PRN number is always needed for OSNMA processing, additional code had to be modified. The signature of the function `split_page` was changed to accept

[1] https://www.gnuradio.org/

the PRN, as well as its invocation in the telemetry decoder in `/src/algorithms/telemetry_decoder/gnuradio_blocks/galileo_telemetry_decoder_gs.cc`. The telemetry decoder always knows the PRN of the satellite, as it obtains it when demodulating the signal, but the `split_page` function learns it only after receiving the navigational word of type 4.

3.1.2 Galmon data feed parser

The Galmon community project [15], provides a freely available stream of navigational data at `86.82.68.237:10000` [2] from a network of GNSS receivers around the world. It provides the data in Protobuf[3] frames.

The script `tools/galmon_to_csv.py` was written to connect to this stream, retrieve the data necessary for OSNMA processing, and output it in the CSV format from before. It uses the protocol definition provided by Galmon (in `navmon.proto` of [15]). The idea for this tool was inspired by the use of the Galmon stream in [16].

3.2 Python OSNMA parser/verifier

An OSNMA data parser and verifier, was implemented in Python, bearing the name *py-osnma-parser*(available on ETH's DINFK GitLab). PyCryptodome [17] was used as the provider of cryptographic primitives. The program reads Galileo nominal pages from a CSV file, reconstructs them into sub-frames, parses OSNMA and navigational data, and performs the various type of verification associated with OSNMA data. Due to the lack of test data, some verification functions and OSNMA states were not implemented, but more on that in chapter 5. The purpose of the program is to authenticate the recorded samples, so the time synchronization requirements between the receiver's time and GST mentioned in subsection 2.5.4 are not explicitly implemented. Data retrieval and verification steps are implemented as described in section 2.5 (summarized from [10, 11]).

The program has 4 main components:

- `osnma.py` - Implements the `OSNMA` class which parses Galileo OSNMA data for a given sub-frame and stores the navigational data associated with it.
- `osnma_storage.py` - Implements the `OSNMA_Storage` class, which stores `OSNMA` objects and reconstructs the multi-part DSM messages
- `osnma_verifier.py` - Implements the `OSNMA_Verifier` class, which implements the verification functions for DSM-KROOT, MACKLT, MACSEQ, TESLA Chain and MAC. The TESLA `Chain` class is also implemented here, which is simply a storage class for the TESLA chain parameters and keys.
- `main.py` - Implements the executable that parses the data from the CSV file, invokes the parsing functionality, and orchestrates the verification steps.

The main script performs the following steps, as outlined in the sequence chart in Figure 3.1:

1. Reads the CSV file from the file name provided as an argument and groups nominal pages by PRN. The nominal pages in each group are in time-ascending order
2. For each group, detects completed sub-frames and extracts the complete 600 bits of OSNMA data and 1920 bits of navigational data from them
3. `OSNMA` objects are constructed from the OSNMA and navigational data for each sub-frame from the previous step, and are added to the `OSNMA_Storage`.

[2] https://twitter.com/GalileoSats/status/1501658640968781825
[3] https://developers.google.com/protocol-buffers

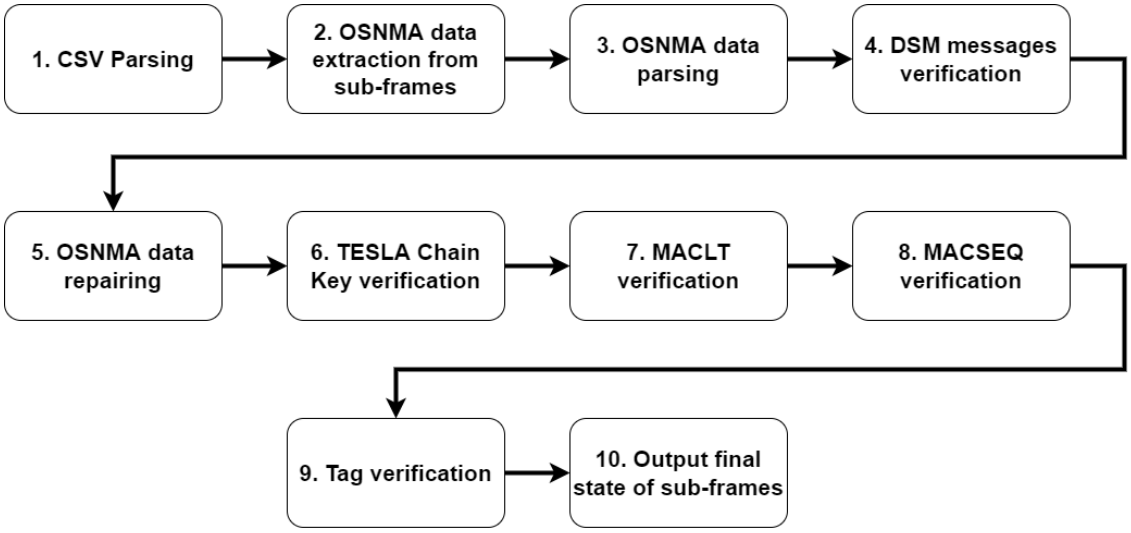

Figure 3.1: Sequence chart of py-osnma-parser

4. Completed DSMs are retrieved from the storage and are forwarded to the DSM-KROOT verification function of the `OSNMA_Verifier`
5. Now that chain parameters are known, the script attempts to re-evaluate the MACK message section of sub-frames which arrived before said parameters were know
6. All sub-frames and the TESLA keys found in them are provided as input to the TESLA chain verification function of the `OSNMA_Verifier`
7. All sub-frames and the MACK messages in them are provided as input to the MAC Look-up Table verification function of the `OSNMA_Verifier`
8. All sub-frames and the MACK messages in them are provided as input to the MACSEQ verification of the `OSNMA_Verifier`
9. All sub-frames, the MACK messages and the navigational data in them are provided as input to the MACSEQ verification of the `OSNMA_Verifier`
10. The program iterates over the OSNMA storage and outputs the state of each sub-frame - whether it was verified, what types of verification it has, whether the MACLT/MACSEQ/TESLA key in it is authentic, etc.

Information on how to use *py-osnma-parser* and the tools from the previous section are available in the README of the project, supplied here in Appendix A.

Chapter 4

Results and evaluation

4.1 Experimental setup

Sample files were retrieved from both the modified version of GNSS-SDR and the `galmon_to_csv` tool. Consult Appendix A for the commands used to collect and process the samples.

The hardware setup to capture OSNMA samples with GNSS-SDR is shown in Figure 4.1. It consists of a (1) HackRF One serving as the RF front-end, using an external clock supplied by a (2) temperature compensated crystal oscillator (TCXO) module. It is connected to a (3) bias tee, which in turn powers the amplifier in the (4) active GNSS antenna.

Unfortunately, even when physical conditions were favorable (the antenna had an unobstructed view of a clear sky and multiple Galileo satellites above the horizon according to the Live World Map of Satellite positions[1]), obtaining samples was challenging due to frequent loss-of-lock events, which occur when the GNSS receiver no longer tracks the signal accurately. In theory, this most often happens when the clock in the receiver is not stable. Other clock sources (OctoClock by Ettus Research) and RF front-ends (USRP B210 by Ettus Research) were tried, but without improvement, likely implying that the problem lies within GNSS-SDR itself.

The modified version of GNSS-SDR was not used to capture OSNMA data directly, as it experienced an even more frequent loss-of-lock, with no apparent logical explanation. As a workaround, the official version of GNSS-SDR was used to dump the raw RF signal to a file, which was then passed as input to the modified version. These raw signal files were large in size, approx. 2 GB per minute of capture, which in turn limited the length of the OSNMA sample.

Retrieving samples with the `galmon_to_csv` is much simpler - only a device with a Python interpreter and connected to the internet is needed. This tool also allows for larger OSNMA samples, as the data is dumped directly into the required CSV format. The data stream for the Galmon network however appeared inconsistent, with nominal pages often being skipped, resulting in incomplete sub-frames, which the parser ignores by design, as the OSNMA data in them is also incomplete.

4.2 Functionality

Testing was performed by running the samples using the setup from the previous section. The program successfully implements OSNMA parsing, DSM-KROOT, TESLA key, MACLT, MACSEQ and Tag verification. In this state, it successfully authenticates navigation data

[1] https://in-the-sky.org/satmap_radar.php

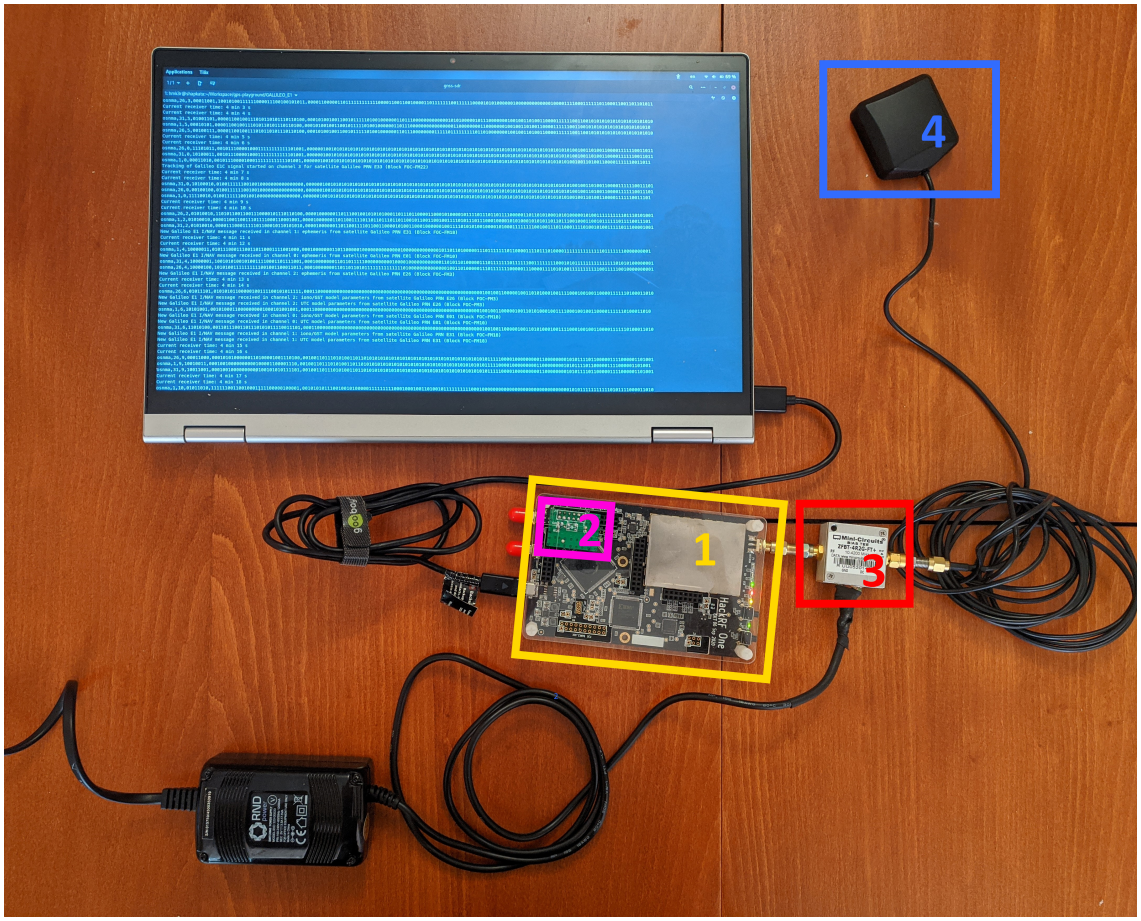

Figure 4.1: Example hardware setup for capturing GNSS signals with GNSS-SDR, using a HackRF One for the RF front-end

under nominal test-phase OSNMA operation. In the following sections, example outputs of the program are presented and described.

4.2.1 OSNMA data parsing

In Figure 4.2, the initial state of the sub-frame can be seen. At this stage, none of the OSNMA information in the frame is verified, which is reflected in the output.

4.2.2 DSM-KROOT verification

The output of the verification of two DSM-KROOT messages is presented in Figure 4.3. M is the message over which the signature was computed. Part of M, naturally, is the Root key (can be seen as the last 16 bytes of M). Note that the two verifications verify different root keys - this is because the sample in question captured a change in the TESLA root key (new "floating key").

4.2.3 TESLA Key verification

In Figure 4.4 and Figure 4.5 the verification process of the TESLA Key can be seen. Each key is derived from the one received after it. In Figure 4.4, the procedure of short-cutting can be seen - once `KEY 051` is confirmed to be authentic, i.e. the root key can be reached through it, if `KEY 052` derives to it then `KEY 052` must also reach the root key.

```
-> PRN: 26
  -> WN: 1196
  -> TOW: 567900
  -> GST_SF: WN: 1196, TOW: 567900
  -> HKROOT:
    -> NMA Status: Test
    -> Chain Status: Nominal
    -> Chain ID: 1
    -> Reserved: 0
    -> DSM ID: 5
    -> DSM Block ID: 1
  -> MACK messages:
    -> Tag_0: 0xd4d385c32b
    -> MACSEQ: 011100000101, NOT Verified, withOUT verified TESLA Key
    -> Reserved: 0001
    -> Tag&Info
      -> Tag: 0x7546a1375c; Info - PRN_D: 25, ADKD: 0, Reserved: 0000
      -> Tag: 0x92f06da2cb; Info - PRN_D: 255, ADKD: 4, Reserved: 0000
      -> Tag: 0xa6bb78f3f8; Info - PRN_D: 14, ADKD: 0, Reserved: 0100
      -> Tag: 0x93126d2c1; Info - PRN_D: 26, ADKD: 12, Reserved: 0001
      -> Tag: 0x2a6d4b10ce; Info - PRN_D: 1, ADKD: 0, Reserved: 0000
    -> TESLA Key: 0xaf8739ba4c9737fc3d52f09cfb2fac62, NOT verified
    -> Padding: 0000000000000000
    -> Subframe MACK Sequence NOT Verified
  -> Navdata Authentic: False
```

Figure 4.2: Result of OSNMA data parsing

```
DSM-KROOT Verification:

M(232 bits): 0x5250492104ac9d22967a3858080ee845b0a91d932e27e882114457575e
Sig(512 bits): 0xa02d253c0ce3650e39d5c8e4b1927804d1094301f1cd27337b5777f11...
Root key(128 bits): 0x0ee845b0a91d932e27e882114457575e
Pad(88 bits): 0x4bd0dd06091e26a276fbac
T(88 bits): 0x4bd0dd06091e26a276fbac
Pad == T? True
Signature Correct? True
DSM-KROOT Verification:

M(232 bits): 0x5250492104ac9e22967a3858088afad0590da613e6df39baea104f5598
Sig(512 bits): 0x108ccad824a09ad0c32136fd4ed1ea243e69cd97f4ae71bc052fd4826e...
Root key(128 bits): 0x8afad0590da613e6df39baea104f5598
Pad(88 bits): 0x99604d61ccbce0220d4468
T(88 bits): 0x99604d61ccbce0220d4468
Pad == T? True
Signature Correct? True
```

Figure 4.3: Result of DSM-KROOT verification

The state of the sub-frame in Figure 4.2 is updated and can be seen in Figure 4.6 - notice that the "TESLA Key" field now marks the key as "verified".

```
TESLA Key Verification:

Key index: 51
CID: 1
KEY 050: 715f1725d4b598321a815b79d6084a3649a61eae25d3964da3a2, b5a2c37e6ecf1d5f4c7c4f31c9892405 =? d499ade8332a073116f53228bf1ef684, WN: 1178, TOW: 401070
KEY 049: b5a2c37e6ecf1d5f4c7c4f31c989240549a61e9025d3964da3a2, 12670e25e398364722d1acb00ad73789 =? d499ade8332a073116f53228bf1ef684, WN: 1178, TOW: 401040
KEY 048: 12670e25e398364722d1acb00ad7378949a61e7225d3964da3a2, ac70e275e9afe92b05851a647f532372 =? d499ade8332a073116f53228bf1ef684, WN: 1178, TOW: 401010
....
KEY 001: 12e88a4fe936fe3c564e76f9b8c7c66b49a618f025d3964da3a2, da479efe4d3feb764e5517aa76752748 =? d499ade8332a073116f53228bf1ef684, WN: 1178, TOW: 399600
KEY 000: da479efe4d3feb764e5517aa7675274849a618d225d3964da3a2, d499ade8332a073116f53228bf1ef684 =? d499ade8332a073116f53228bf1ef684, WN: 1178, TOW: 399570
Chained reached a key that was verified; short-cutting
Key index: 52
CID: 1
KEY 051: 6d07fd919449ee2b1e66629c92fb6b4d49a61ecc25d3964da3a2, 715f1725d4b598321a815b79d6084a36 =? d499ade8332a073116f53228bf1ef684, WN: 1178, TOW: 401100
Chained reached a key that was verified; short-cutting
Key index: 53
CID: 1
KEY 052: f85b25d820147d1cc5508480c6d524cc49a61eea25d3964da3a2, 6d07fd919449ee2b1e66629c92fb6b4d =? d499ade8332a073116f53228bf1ef684, WN: 1178, TOW: 401130
Chained reached a key that was verified; short-cutting
Key index: 54
CID: 1
KEY 053: fc97b6fd8fddeff394c1a1edb4f8823449a61f0825d3964da3a2, f85b25d820147d1cc5508480c6d524cc =? d499ade8332a073116f53228bf1ef684, WN: 1178, TOW: 401160
```

Figure 4.4: Result of TESLA Key verification

```
TESLA Key Verification:

Key is associated with older (floating) HKROOT at GST_SF_K WN: 1196, TOW: 565230
Key index: 71
CID: 1
KEY 070: 8309eb4a645ee1fbe875fc80b3f5299f4ac8a7e622967a385808, 5b65b3680b2935fc2583fb80ad957009 =? 0ee845b0a91d932e27e882114457575e, WN: 1196, TOW: 567270
KEY 069: 5b65b3680b2935fc2583fb80ad9570094ac8a7c822967a385808, 1b3fb6336ba07c88c4c61dd9a1e36372 =? 0ee845b0a91d932e27e882114457575e, WN: 1196, TOW: 567240
KEY 068: 1b3fb6336ba07c88c4c61dd9a1e363724ac8a7aa22967a385808, 136f0fdd82cee8690f624077dea0e564 =? 0ee845b0a91d932e27e882114457575e, WN: 1196, TOW: 567210
...
KEY 001: 9b84b50e655bb390ddf8bf9833a058214ac89fd022967a385808, 8c897fb743347efb80a9b2a9757f3c6b =? 0ee845b0a91d932e27e882114457575e, WN: 1196, TOW: 565200
KEY 000: 8c897fb743347efb80a9b2a9757f3c6b4ac89fb222967a385808, 0ee845b0a91d932e27e882114457575e =? 0ee845b0a91d932e27e882114457575e, WN: 1196, TOW: 565170
Key is associated with older (floating) HKROOT at GST_SF_K WN: 1196, TOW: 565230
Key index: 72
CID: 1
KEY 071: 339f386272e928c0906175e98de43b164ac8a80422967a385808, 8309eb4a645ee1fbe875fc80b3f5299f =? 0ee845b0a91d932e27e882114457575e, WN: 1196, TOW: 567300
KEY 070: 8309eb4a645ee1fbe875fc80b3f5299f4ac8a7e622967a385808, 5b65b3680b2935fc2583fb80ad957009 =? 0ee845b0a91d932e27e882114457575e, WN: 1196, TOW: 567270
KEY 069: 5b65b3680b2935fc2583fb80ad9570094ac8a7c822967a385808, 1b3fb6336ba07c88c4c61dd9a1e36372 =? 0ee845b0a91d932e27e882114457575e, WN: 1196, TOW: 567240
KEY 068: 1b3fb6336ba07c88c4c61dd9a1e363724ac8a7aa22967a385808, 136f0fdd82cee8690f624077dea0e564 =? 0ee845b0a91d932e27e882114457575e, WN: 1196, TOW: 567210
KEY 067: 136f0fdd82cee8690f624077dea0e5644ac8a78c22967a385808, f52a89b6c0e6c69efdc16d22b8cc6745 =? 0ee845b0a91d932e27e882114457575e, WN: 1196, TOW: 567180
KEY 066: f52a89b6c0e6c69efdc16d22b8cc67454ac8a76e22967a385808, 0fca26d06a6dbd0812947e64beb538ec =? 0ee845b0a91d932e27e882114457575e, WN: 1196, TOW: 567150
KEY 065: 0fca26d06a6dbd0812947e64beb538ec4ac8a75022967a385808, 51026ce1605e65c7e5c28fa088d15e0c =? 0ee845b0a91d932e27e882114457575e, WN: 1196, TOW: 567120
KEY 064: 51026ce1605e65c7e5c28fa088d15e0c4ac8a73222967a385808, c88a198f9ab382a9b788f3b515c77af4 =? 0ee845b0a91d932e27e882114457575e, WN: 1196, TOW: 567090
KEY 063: c88a198f9ab382a9b788f3b515c77af44ac8a71422967a385808, 09221932cd7ffe3d0a4dc4136f404ec9 =? 0ee845b0a91d932e27e882114457575e, WN: 1196, TOW: 567060
KEY 062: 09221932cd7ffe3d0a4dc4136f404ec94ac8a6f622967a385808, ed5e9a73406ad79e94c062f9563f2410 =? 0ee845b0a91d932e27e882114457575e, WN: 1196, TOW: 567030
KEY 061: ed5e9a73406ad79e94c062f9563f24104ac8a6d822967a385808, 79349a9259978cd839dd3fd86c6c94c9 =? 0ee845b0a91d932e27e882114457575e, WN: 1196, TOW: 567000
```

Figure 4.5: Result of TESLA Key verification with floating KROOT

4.2.4 MACSEQ and MACLT verification

Figure 4.7 shows the output of the MACSEQ verification. The comparison between the computed value and the one provided by the satellite can be seen, as well as the GST and PRN of the sub-frame over which the tag was computed. MACLT verification is a trivial string comparison operation that does not involve cryptographic primitives, so detailed steps of it are not shown. The state of the sub-frame from Figure 4.6 is now updated to what can be seen in Figure 4.8 - the "MACSEQ" field is now marked as verified and the Sub-frame MACK Sequence is marked as matching the one in the MAC Look-up table.

4.2.5 Tag verification

The tag verification process can be seen in Figure 4.9 - various tags are computed for different types of navdata authentication. The GST time of the sub-frame containing the key is displayed, along with the PRN of the satellite providing the OSNMA data, the type of authentication, and the computed and received (reference) tags. This step produces the final state

```
-> PRN: 26
  -> WN: 1196
  -> TOW: 567900
  -> GST_SF: WN: 1196, TOW: 567900
  -> HKROOT:
    -> NMA Status: Test
    -> Chain Status: Nominal
    -> Chain ID: 1
    -> Reserved: 0
    -> DSM ID: 5
    -> DSM Block ID: 1
  -> MACK messages:
    -> Tag_0: 0xd4d385c32b
    -> MACSEQ: 011100000101, NOT Verified, withOUT verified TESLA Key
    -> Reserved: 0001
    -> Tag&Info
      -> Tag: 0x7546a1375c; Info - PRN_D: 25, ADKD: 0, Reserved: 0000
      -> Tag: 0x92f06da2cb; Info - PRN_D: 255, ADKD: 4, Reserved: 0000
      -> Tag: 0xa6bb78f3f8; Info - PRN_D: 14, ADKD: 0, Reserved: 0100
      -> Tag: 0x93126d2c1; Info - PRN_D: 26, ADKD: 12, Reserved: 0001
      -> Tag: 0x2a6d4b10ce; Info - PRN_D: 1, ADKD: 0, Reserved: 0000
    -> TESLA Key: 0xaf8739ba4c9737fc3d52f09cfb2fac62, verified
    -> Padding: 0000000000000000
    -> Subframe MACK Sequence NOT Verified
  -> Navdata Authentic: False
```

Figure 4.6: Sub-frame state after TESLA Key verification

of a sub-frame, which can be seen and is described in the next section. Note that some sub-frames are missing, specifically for cross-authentication - the OSNMA data capture methods implemented unfortunately manage to capture only a few satellites at a time, and this subset is not enough to perform cross-authentication.

4.2.6 Final output for a sub-frame

Figure 4.10 represents the final state of a sub-frame, for which all the OSNMA data within it has been verified. Comparing it to Figure 4.2, the MACSEQ, TESLA Key, MACK Sequence are now verified. The navigational data within the sub-frame has also been verified, with the different types of authentications listed.

4.3 Time necessary to process OSNMA

The time necessary to process OSNMA messages is naturally bound by the time needed to receive the sub-frame from the satellite - 30 seconds. In the various verification steps, there is usually only a single invocation of a cryptographic primitive, so benchmarking the verification steps will result in benchmarking the underlying cryptographic primitives, which would be redundant, as these performance measurements are already well known.

An important time metric is, however, the time necessary to begin OSNMA Tag verification with certainty, i.e. the time needed to receive the root key of the TESLA chain and verify its signature. This time depends on the TESLA key size and the signature scheme used. In the

```
MACSEQ Verification:

PRN: 26, WN: 1196, TOW: 567300; 001011000000 =? 001011000000 - True
PRN: 26, WN: 1196, TOW: 567330; 101001110001 =? 101001110001 - True
PRN: 26, WN: 1196, TOW: 567360; 111011110100 =? 111011110100 - True
PRN: 26, WN: 1196, TOW: 567390; 010011011001 =? 010011011001 - True
PRN: 26, WN: 1196, TOW: 567420; 111111000101 =? 111111000101 - True
PRN: 26, WN: 1196, TOW: 567450; 101011001110 =? 101011001110 - True
PRN: 26, WN: 1196, TOW: 567480; 000111110010 =? 000111110010 - True
PRN: 26, WN: 1196, TOW: 567510; 010010001011 =? 010010001011 - True
PRN: 26, WN: 1196, TOW: 567540; 111101000010 =? 111101000010 - True
PRN: 26, WN: 1196, TOW: 567570; 001111111011 =? 001111111011 - True
PRN: 26, WN: 1196, TOW: 567600; 110101101110 =? 110101101110 - True
PRN: 26, WN: 1196, TOW: 567630; 101010110101 =? 101010110101 - True
PRN: 26, WN: 1196, TOW: 567660; 111110010011 =? 111110010011 - True
PRN: 26, WN: 1196, TOW: 567690; 011001101011 =? 011001101011 - True
PRN: 26, WN: 1196, TOW: 567720; 110111001010 =? 110111001010 - True
PRN: 26, WN: 1196, TOW: 567750; 100111000010 =? 100111000010 - True
PRN: 26, WN: 1196, TOW: 567780; 001001011010 =? 001001011010 - True
PRN: 26, WN: 1196, TOW: 567810; 000010111001 =? 000010111001 - True
PRN: 26, WN: 1196, TOW: 567840; 001000010101 =? 001000010101 - True
PRN: 26, WN: 1196, TOW: 567870; 110001111100 =? 110001111100 - True
PRN: 26, WN: 1196, TOW: 567900; 011100000101 =? 011100000101 - True
PRN: 26, WN: 1196, TOW: 567930; 001010101011 =? 001010101011 - True
PRN: 26, WN: 1196, TOW: 567960; 101000001101 =? 101000001101 - True
```

Figure 4.7: Result of MACSEQ verification

current OSNMA test-phase configuration, which can be found in [11], 8 DSM blocks must be received to retrieve the KROOT and its signature. As each sub-frame(of length 30 seconds) contains a single DSM block, $8 * 30 = 240$ `seconds` $= 4$ `minutes` are needed. This was also observed in practice.

```
-> PRN: 26
  -> WN: 1196
  -> TOW: 567900
  -> GST_SF: WN: 1196, TOW: 567900
  -> HKROOT:
    -> NMA Status: Test
    -> Chain Status: Nominal
    -> Chain ID: 1
    -> Reserved: 0
    -> DSM ID: 5
    -> DSM Block ID: 1
  -> MACK messages:
    -> Tag_0: 0xd4d385c32b
    -> MACSEQ: 011100000101, Verified, with verified TESLA Key
    -> Reserved: 0001
    -> Tag&Info
      -> Tag: 0x7546a1375c; Info - PRN_D: 25, ADKD: 0, Reserved: 0000
      -> Tag: 0x92f06da2cb; Info - PRN_D: 255, ADKD: 4, Reserved: 0000
      -> Tag: 0xa6bb78f3f8; Info - PRN_D: 14, ADKD: 0, Reserved: 0100
      -> Tag: 0x93126d2c1; Info - PRN_D: 26, ADKD: 12, Reserved: 0001
      -> Tag: 0x2a6d4b10ce; Info - PRN_D: 1, ADKD: 0, Reserved: 0000
    -> TESLA Key: 0xaf8739ba4c9737fc3d52f09cfb2fac62, verified
    -> Padding: 0000000000000000
    -> Subframe MACK Sequence Verified
  -> Navdata Authentic: False
```

Figure 4.8: Sub-frame state after MACLT and MACSEQ verification

```
WN: 1196, TOW: 569940, Auth 7, Self-12 - Computed tag: 0dd9982d7e, Reference: 0dd9982d7e Equal? - True
WN: 1196, TOW: 569940, Auth 7, Self-Tag0 - Computed tag: f60f4a3935, Reference: f60f4a3935 Equal? - True
No subframe from PRN 9 at time WN: 1196, TOW: 569910 for Cross-0 authentication
No subframe from PRN 5 at time WN: 1196, TOW: 569910 for Cross-0 authentication
No subframe from PRN 33 at time WN: 1196, TOW: 569910 for Cross-0 authentication
WN: 1196, TOW: 569940, Auth 1, Self-12 - Computed tag: 39ab0ef388, Reference: 39ab0ef388 Equal? - True
WN: 1196, TOW: 569940, Auth 1, Self-Tag0 - Computed tag: 8a5a71d1eb, Reference: 8a5a71d1eb Equal? - True
No subframe from PRN 25 at time WN: 1196, TOW: 569910 for Cross-0 authentication
No subframe from PRN 27 at time WN: 1196, TOW: 569910 for Cross-0 authentication
No subframe from PRN 33 at time WN: 1196, TOW: 569910 for Cross-0 authentication
WN: 1196, TOW: 569940, Auth 12, Self-12 - Computed tag: dfa111a33b, Reference: dfa111a33b Equal? - True
WN: 1196, TOW: 569940, Auth 12, Self-Tag0 - Computed tag: f7de1dda6c, Reference: f7de1dda6c Equal? - True
No subframe from PRN 33 at time WN: 1196, TOW: 569910 for Cross-0 authentication
No subframe from PRN 25 at time WN: 1196, TOW: 569910 for Cross-0 authentication
No subframe from PRN 9 at time WN: 1196, TOW: 569910 for Cross-0 authentication
WN: 1196, TOW: 569940, Auth 4, Self-12 - Computed tag: bca1d934ba, Reference: bca1d934ba Equal? - True
WN: 1196, TOW: 569940, Auth 4, Self-Tag0 - Computed tag: 0133d32d74, Reference: 0133d32d74 Equal? - True
No subframe from PRN 27 at time WN: 1196, TOW: 569910 for Cross-0 authentication
No subframe from PRN 19 at time WN: 1196, TOW: 569910 for Cross-0 authentication
No subframe from PRN 25 at time WN: 1196, TOW: 569910 for Cross-0 authentication
WN: 1196, TOW: 569940, Auth 2, Self-12 - Computed tag: bd8f9867bc, Reference: bd8f9867bc Equal? - True
WN: 1196, TOW: 569940, Auth 2, Self-Tag0 - Computed tag: 0086c965d7, Reference: 0086c965d7 Equal? - True
```

Figure 4.9: Result of TAG verification

```
-> PRN: 26
  -> WN: 1196
  -> TOW: 567900
  -> GST_SF: WN: 1196, TOW: 567900
  -> HKROOT:
    -> NMA Status: Test
    -> Chain Status: Nominal
    -> Chain ID: 1
    -> Reserved: 0
    -> DSM ID: 5
    -> DSM Block ID: 1
  -> MACK messages:
    -> Tag_0: 0xd4d385c32b
    -> MACSEQ: 011100000101, Verified, with verified TESLA Key
    -> Reserved: 0001
    -> Tag&Info
      -> Tag: 0x7546a1375c; Info - PRN_D: 25, ADKD: 0, Reserved: 0000
      -> Tag: 0x92f06da2cb; Info - PRN_D: 255, ADKD: 4, Reserved: 0000
      -> Tag: 0xa6bb78f3f8; Info - PRN_D: 14, ADKD: 0, Reserved: 0100
      -> Tag: 0x93126d2c1; Info - PRN_D: 26, ADKD: 12, Reserved: 0001
      -> Tag: 0x2a6d4b10ce; Info - PRN_D: 1, ADKD: 0, Reserved: 0000
    -> TESLA Key: 0xaf8739ba4c9737fc3d52f09cfb2fac62, verified
    -> Padding: 0000000000000000
    -> Subframe MACK Sequence Verified
  -> Navdata Authentic: True
    -> Tag Type: Self-Tag0, Verified by PRN 26, with verified TESLA Key
    -> Tag Type: Self-12, Verified by PRN 26, with verified TESLA Key
```

Figure 4.10: Final output for a sub-frame

Chapter 5

Discussion

It is clear that the authentication of navigational data is possible and OSNMA presents itself as a fairly elegant solution. While the specification documents for OSNMA [10, 11] give enough technical information to make its implementation fairly straightforward, there is a bigger issue that hinders development: the lack of easily accessible test data. The only test data provided officially by Galileo's operators can be found in [11] - the already collected and parsed DSM-KROOT, DSM-PKR and other messages. While some of this data can be used for unit testing, an end-to-end test which includes collecting the multipart messages from separate pages and parsing them is rather impossible to reverse-engineer from the provided data. An added annoyance is the fact that a developer has to copy-paste this information from a not-perfectly formatted PDF file. For some events, like chain revocations and public key changes, in which concepts like time of applicability play a key role, a continuous data sample is absolutely mandatory so that the functionality can be tested fully. Normally, a developer could simply resort to recording the actual data coming from satellites and use that as test input. Unfortunately, in the test phase of the system, only DSM-KROOT and MACK messages have been retrieved so far, and other messages such as chain revocations, DSM-PKR for public-key renewals, have not been broadcast at all. Even with DSM-KROOT and MACK messages, only a single permutation of the possible parameters has been presented. It is therefore almost imperative that Galileo's operator provides a tool for developers to generate OSNMA messages of their choosing(naturally not including the production keys), or at least publishes a more complete collection of test data. During the test phase, an ideal solution is to instruct the OSNMA "controller" to iterate over possible OSNMA configurations and broadcast rare messages like DSM-PKR and chain revocations regularly, so that GNSS receiver developers can test their implementations in practice.

5.1 Possible attacks and countermeasures

While OSNMA is a step in the right direction towards more secure navigational services, it is by no means a perfect solution. While implementing the protocol, some issues with the design presented themselves based on which attacks against OSNMA could be possible. A deeper and detailed exploration of these attacks lies out of the scope of this project, so only basic ideas are presented, along with some trivial countermeasures. When discussing countermeasures, internet connectivity is considered available, as it is more or less part of the OSNMA requirements, as described in subsection 2.5.4

5.1.1 Downgrade attacks

Per the OSNMA specification [10], not all satellites are transmitting OSNMA data, i.e. the OSNMA field is zeroed out, and receivers should therefore not process the field supplied by such satellites. This presents a possible attack vector, where a malicious actor zeros out the field and supplies navigational data of their choosing. OSNMA supports the authentication of navigational data of such satellites through cross-authentication - other satellites transmit the associated TESLA keys and tags, computed over the navigational data of the OSNMA-disabled ones. However, the attacker could simply zero out the OSNMA field of all satellites, leading to no OSNMA processing taking place.

Countermeasures based on signal characteristics will likely thwart such an attack, as they would naturally identify the attacker's signal as non-genuine. Receivers which are able to connect to the internet could implement a more straightforward detection mechanism - they could consult with a server on whether OSNMA is enabled for a specific satellite, and if their measurements differ from those retrieved from the server, they could temporarily blacklist the misbehaving satellite(s). Although Galileo's Service Centre does not provide such a service at the time of writing, the website of the Galmon community project[1], through its network of receivers around the world, provides observational information on whether OSNMA is enabled for a particular satellite.

5.1.2 Relay attacks

OSNMA guarantees that navigational messages are relatively "fresh", with regard to the time synchronization requirements, making detecting replay attacks possible (attacks in which old navigational messages are retransmitted by the attacker). However, relay attacks, which were briefly explained in subsection 2.1.2, still pose a threat. As the navigational data and OSNMA data in the replayed messages is inherently authentic, the OSNMA processor would fail to detect the attack. As the time synchronization requirement of OSNMA is fairly loose(T_L of 30 to 330 seconds), the signal could in theory be recorded at any point on earth, quickly sent through the internet, and then retransmitted at the location of the victim. During the time of writing this report, Motallebighomi et al. [18], presented a paper that implements this type of attack, with the proof-of-concept yielding promising results. Detection mechanisms based on signal characteristics would probably give the best chance to observe this attack.

5.1.3 Short-term spoofing

Spoofing during normal operation

As described in the OSNMA specification, navigational data is authenticated typically 1 minute after being received - SF_T contains the data, SF_{T+1} contains the MAC tags, and SF_{T+2} contains the associated TESLA key, where SF_T is the 30-second long sub-frame transmitted at time T. This effectively means that OSNMA provides "retrospective" authentication. If the receiver chooses to interpret the navigational data before it is authenticated, for the sake of being more accurate, problems could arise. If an attacker modifies only the navigational data, while keeping OSNMA data intact, the attack would go unnoticed in this 1-minute interval, which, for an example, in an urban are might be enough to detour an autonomous car a few blocks away. It is important that the MAC tags and TESLA key transmitted alongside the spoofed navigational data are authentic; to better understand this, consider the following scenarios:

1. Spoofing navdata in SF_{T+1} - if the attacker supplies invalid MAC tags, SF_T will be marked as unauthentic when the TESLA key in SF_{T+2} arrives, which still gives the attacker 30 seconds of spoofed location time

[1] https://galmon.eu/

2. Spoofing navdata in SF_{T+2} - if the attacker supplies an invalid TESLA key, SF_T will be marked as unauthentic instantly, leading to an immediate detection of an attack.

The main complexity of this attack comes from the fact that the OSNMA field contains unpredictable data, such as tags and TESLA keys, which, in turn, make the CRC code unpredictable. If an attacker modifies CRC-protected fields, such as the navigational data field, without manipulating the CRC field accordingly, CRC check fails and the page will be ignored by the receiver. To "repair" the CRC field when modifying the OSNMA data, a technique similar to what is presented in [19] or attacks such as early-detect/late-commit (ED/LC) [20, 21] might be employed.

A countermeasure to this attack would be for the receiver to include the navdata in PVT computations only after it is verified. This could introduce PVT inaccuracies in receivers which are not well synchronized to GST and have to use only slow MAC (for these, 6 minutes are needed to verify data, because of the 10-sub-frame long key delay). Another countermeasure, which reduces the time needed to detect the attack, is to simply broadcast the MAC tags alongside the navigational data they were computed over, i.e. SF_T contains the data and its MAC tags, SF_{T+1} contains the TESLA key associated with them. This reduces the time necessary for verification from 1 minute to 30 seconds. Unfortunately, this also contradicts the original TESLA specification on determining the key disclosure delay (section 3.6 of the RFC [12]), which requires at least 2 time intervals between the disclosure of the MAC and the key to combat issues imposed by the propagation delay and the receiver time synchronization error. However, given the large time interval of 30 seconds, these issues might not be present in OSNMA.

Cold start attack

On a cold start, when no previous verified DSM-KROOTs are available to the receiver, $\approx$4 minutes (in the current test configuration) are needed to receive the initial data needed for OSNMA verification - the TESLA chain parameters, the KROOT and its digital signature. The receiver can begin verifying tags as soon as it receives the chain parameters, which are needed to parse the OSNMA message, but those tags would not be completely verified, as the TESLA keys used to generate them cannot be proven to be authentic before the KROOT and its DSM arrive. During this time window, an attacker could spoof navigational data(naturally, the attacker should be careful with the CRC code), as a receiver could have been programmed to accept data deemed unauthentic during this period to retain availability. Depending on the implementation of the OSNMA processor, while KROOT is not available, instead of checking whether a TESLA key can be used to derive it, verify temporal authenticity, that is, the previously received TESLA key K_N can be derived from the new key K_{N+1}. This makes the attack a bit more complicated, but not impossible, as the attacker could generate their own TESLA chain, following the parameters supplied by the DSM-KROOT message, compute the tags, and transmit them alongside the keys and the spoofed data. This way, the receiver will be able to verify that the TESLA chain is not authentic only after receiving the Galileo-issued DSM-KROOT.

A viable countermeasure would be to get the navigational data from somewhere else until the KROOT is received and verified. An example work-around is to use a previously retrieved almanac(stored from a previous operation of the receiver or retrieved from GSC's server) for the position calculation. Although the nature of the almanac means that the computed position will not be precisely accurate, it would keep the receiver available. Another possibility is retrieving the KROOT and its signature through the internet, to eliminate OSNMA cold-starts completely, but this option should be explored carefully, as this could reveal some of the "unpredictable" bits in the OSNMA field to the attacker ahead of time, making forging the CRC code easier.

Chapter 6

Future work

Besides the obvious further steps, such as the exploration and development of the attacks described in section 5.1 and the implementation of countermeasures defined therein, the exploration of GNSS and Galileo OSNMA in this project brought other ideas, closely related to the matter at hand.

6.1 Official open-source Galileo OSNMA processing library

As will be seen in chapter 7, many experimental implementations of an OSNMA processor exist. It is likely that even more proprietary ones will appear, written by GNSS receiver manufacturers. This poses a problem, since many different implementations mean many unintentionally different interpretations of the OSNMA protocol, each of which will likely contain different bugs. An ideal solution would be an official open-source OSNMA library, supplied by the European Union Agency for the Space Programme and developed by them and the community. Such a library should provide an interface through which the receiver can:

1. Send the pages received from the satellite to the library
2. Poll the OSNMA processor for verification messages - they would contain information such as what type of verification(DSM-HKROOT, MAC Tag, TESLA chain), whether it was successful, and any associated data (e.g. in the case of MAC Tag verification, the GST and PRN of the sub-frame that was verified)

An idea for the class diagram for such a library can be seen in Figure 6.1. Ideally, this library will be written in C/C++, in order to be cross-platform and able to run on even proprietary microcontrollers/SoCs, such as the ones found on some proprietary GNSS receivers. The library should support many cryptographic back-ends, as not all devices share a common one.

6.2 Implementing Galileo OSNMA in GNSS-SDR

While it is generally not used in mission-critical navigational deployments, GNSS-SDR provides a fairly user-friendly way for developers and researchers to debug GNSS signals, so implementing Galileo OSNMA processing in it is a logical step toward enabling a more practical approach to OSNMA security research.

A proof-of-concept OSNMA processing was implemented in GNSS-SDR, which successfully retrieves OSNMA data, collects DSM messages, and performs DSM-KROOT verification. OSNMA processing is implemented as a GNU Radio block which runs on a separate thread. The blocks which convert the RF signal to bits then send the OSNMA data to this block,

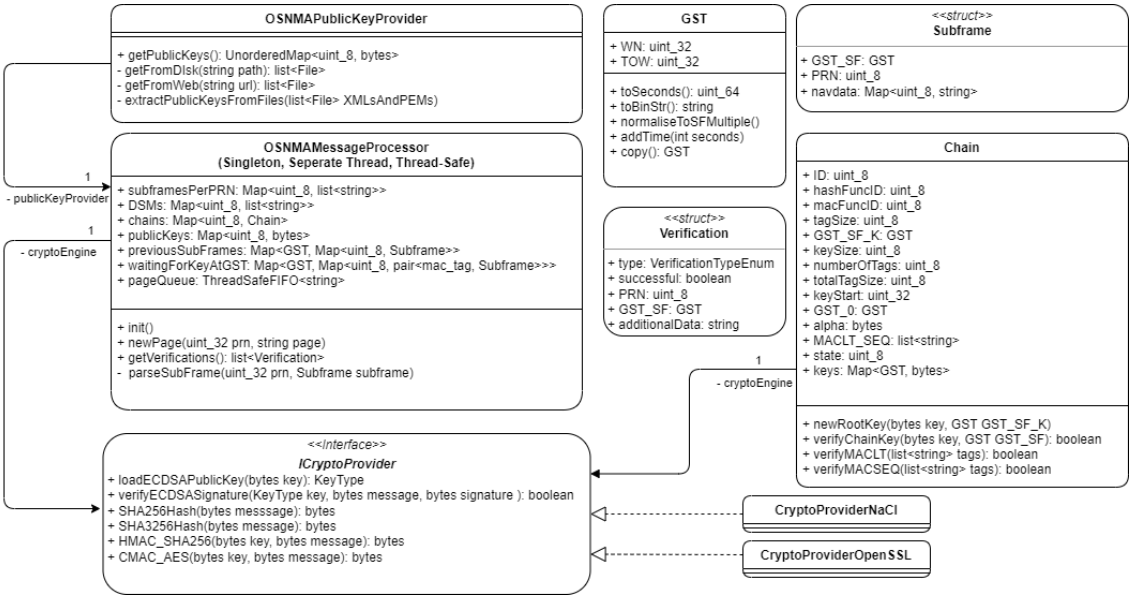

Figure 6.1: A sample UML class-diagram for a OSNMA processing library

through GNU Radio's message passing[1] functionality(each block runs on a separate thread, so message passing was designed to be thread-safe). The OSNMA block then forwards this information to an OSNMA library and polls it for new verification messages. Any new verification messages should then be forwarded to the PVT block(through the message passing mechanism), which it can use to influence position calculation. A block diagram for GNSS-SDR reflecting these changes is presented in Figure 6.2. An example run of GNSS-SDR with DSM-KROOT verification is shown in Figure 6.3

The OSNMA library in the proof-of-concept implementation does not have any dependencies with GNSS-SDR, and in theory can be extracted to a standalone one. While only the functionality mentioned above is implemented in it, it can be used as the skeleton for a fully featured OSNMA library, which is subject to future work. Processing of verification messages by the PVT block also awaits implementation, though the procedure is rather unclear as the Galileo OSNMA specification leaves it up to the individual developer to decide how to handle the results of OSNMA.

The fork of GNSS-SDR with OSNMA processing can be found on ETH's GitLab.

[1] https://wiki.gnuradio.org/index.php/Message_Passing

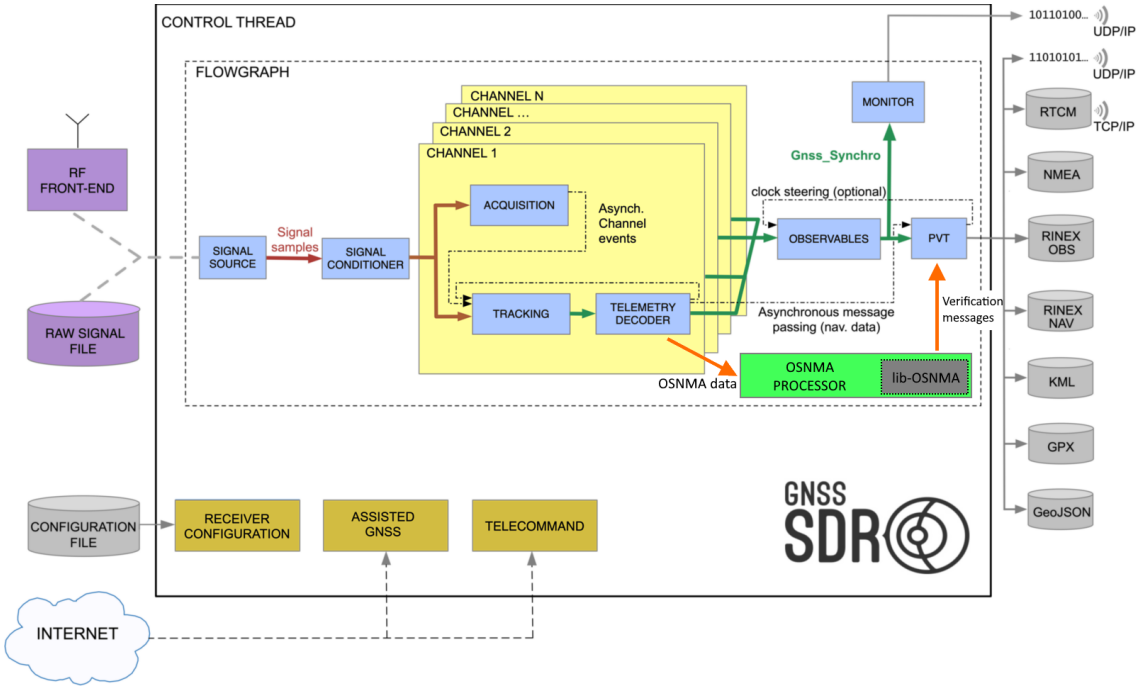

Figure 6.2: The block diagram of GNSS-SDR with OSNMA processing, adapted from [22]

Figure 6.3: Log of DSM-KROOT verification in GNSS-SDR

Chapter 7

Related work

Other projects, such as osnmaPython [23] and galileo-osnma [16], implement OSNMA processing to some extent in different programming languages and support different GNSS receivers. At the time of writing, the library osnmaPython appears to only implement OSNMA message parsing and does not perform the cryptographic verification functions associated with OSNMA. Galileo-OSNMA, on the other hand, is more complete and offers almost the same feature set as the implementation in this paper, but written in Rust and architecturally different. The two mentioned are purely implementations of OSNMA and do not provide a discussion on the security of OSNMA, which is the key difference between them and this project.

A notable implementation of OSNMA is OSNMALib [24], which has been supported by the European Commission and the Spanish Ministry of Science and Innovation. This project implements OSNMA as a Python library, and implements the OSNMA specification almost to its full extent. The paper is focused on the implementation aspect and does provide a discussion on OSNMA, as done here. More notably, the authors of the paper were able to capture public key revocation, new public key, and TESLA chain change samples. These signals were transmitted in late 2020 and early 2021, before the public test phase of OSNMA began (November 2021) and unfortunately before this project was started. The paper and data were published in August 2022, which did not leave enough time to implement and test the functionality enabled by these samples in this project. However, these samples are now publicly available, which should make testing implementations of OSNMA easier.

Chapter 8

Conclusion

With this project, OSNMA is explored as a means of authenticated positioning. A parser for the protocol implemented in Python is provided. Suggestions on how OSNMA processor developers can be aided are made. The idea of a reference implementation of OSNMA in the form of an official general-purpose library is proposed, along with an example architecture. Then, a proof-of-concept is presented for implementing OSNMA in a GNSS receiver software, which also introduces the skeleton of the OSNMA library and demonstrates how it could be used. The results confirm that OSNMA is indeed functional and an advancement in GNSS security, while also proving non-cryptographic protection mechanisms are still necessary.

Appendix A

py-osnma-parser README

py-osnma-parser is a parser and verifier for Galileo OSNMA data

A.1 Capabilities

The parser reads the data from a CSV file, where each row is a Galileo I/NAV nominal page, provided in the format:

```
osnma,[PRN],[Word type],[HKROOT message],[MACK message],[navdata]
```

The verifier supports:

- DSM-KROOT verification
- TESLA Chain Key Verification
- MAC Look-up Table Verification
- MACSEQ Verification
- Tag Verification

This functionality allows for complete verification of navigational data.

Not implemented (due to lack of test data):

- DSM-PKR verification
- TESLA Chain state transitions
- Chain revocation
- Public key revocation

A.2 Usage

Before running, install the dependencies using

```
> pip install -r requirements.txt
```

To run the program:

```
> python3 main.py [samples_file.csv] [osnma_public_keys_dir]
```

- `samples_file.csv` - path to the CSV file with samples; Default: `./data/osnma-capture.csv`

- `osnma_public_keys_dir` - path to the folder containing the public ECDSA OSNMA keys; Default: `./osnma-keys/`

An assortment of CSV sample files is provided in `./data` and example outputs of the program are provided in `./log`

An example invocation of the program would be

```
> python3 main.py data/osnma-galmon-hkroot-change.csv
```

As the output is long, it is recommended to pipe the output to a program like `less`

```
> python3 main.py data/osnma-galmon-hkroot-change.csv | less
```

or save it to a file

```
> python3 main.py data/osnma-galmon-hkroot-change.csv > log/example.log
```

A.3 Obtaining CSV samples

Below you can find instructions on how to retrieve your own OSNMA CSV samples and run them through py-osnma-parser.

A.3.1 Patched version of GNSS-SDR

GNSS-SDR can be patched to output the data in the CSV format required above.

To do so, simply apply the patch (`./tools/gnss-sdr-osnma-log.patch`) on a fresh copy of gnss-sdr

```
> git clone https://github.com/gnss-sdr/gnss-sdr.git
> cd gnss-sdr
> git apply /path/to/gnss-sdr-osnma-log.patch
```

and follow the build instructions in gnss-sdr's README.

The patched version will then output the osnma data to `C++`'s `std::cerr` in the CSV format specified before. To store it, redirect the standard error stream to a file:

```
> ./gnss-sdr --config_file=/path/to/my_receiver.conf 2>/tmp/osnma-data-gnss-sdr.csv
```

Then, run it through the parser:

```
> python3 main.py /tmp/osnma-data-gnss-sdr.csv
```

A.3.2 Galmon data stream

OSNMA data can be retrieved from Galmon's navigational data stream. The tool `galmon_to_csv` (found in `./tools/galmon_to_csv.py`) was developed for this purpose.

To get the data samples, you'll need a (stable) internet connection.

The tool outputs the data to the standard output, in the CSV format specified before. Therefore, samples are recorded in the following way:

```
> python3 tools/galmon_to_csv.py > data/osnma-data-galmon.csv
```

This can be then ran through py-osnma-parser like so:

```
> python3 main.py data/osnma-data-galmon.csv
```

Acronyms

ADKD Authentication Data & Key Delay

CID Chain ID

CPKS Chain and Public Key Status

CREV Chain Revoked

DSM Digital Signature Message

DSM-KROOT DSM for a KROOT

DSM-PKR DSM for a PKR

EUSPA European Union Agency for the Space Programme

GNSS Global Navigation Satellite System

GPS Global Positioning System

GSC European GNSS Service Centre

GST Galileo System Time

HKROOT Header and KROOT

KROOT Root Key

MAC Message Authentication Code

MACK MAC and Key

MACLT MAC Look-up Table

MACSEQ MAC Sequence

NMA Navigation Message Authentication

OSNMA Open Service Navigation Message Authentication

PK Public Key

PKID Public Key ID

PKR Public Key Renewal

PKREV Public Key Revocation

PRN Pseudo-random Noise (code)

PVT Position, Velocity and Time

TESLA Timed Efficient Stream Loss-Tolerant Authentication

TOW Time of Week

WN Week Number

Bibliography

[1] M. L. Psiaki and T. E. Humphreys, "Gnss spoofing and detection," *Proceedings of the IEEE*, vol. 104, no. 6, pp. 1258–1270, 2016.

[2] "Galileo Open Service Navigation Message Authentication (OSNMA) | European GNSS Service Centre." [Online]. Available: https://www.gsc-europa.eu/galileo/services/galileo-open-service-navigation-message-authentication-osnma

[3] P. J. Teunissen and O. Montenbruck, *Springer handbook of global navigation satellite systems*. Springer, 2017, vol. 10.

[4] M. Cuntz, A. Konovaltsev, A. Dreher, and M. Meurer, "Jamming and spoofing in gps/gnss based applications and services – threats and countermeasures," in *Future Security*, N. Aschenbruck, P. Martini, M. Meier, and J. Tölle, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 196–199.

[5] P. Papadimitratos and A. Jovanovic, "Gnss-based positioning: Attacks and countermeasures," in *MILCOM 2008 - 2008 IEEE Military Communications Conference*, 2008, pp. 1–7.

[6] A. J. Kerns, D. P. Shepard, J. A. Bhatti, and T. E. Humphreys, "Unmanned aircraft capture and control via gps spoofing," *Journal of Field Robotics*, vol. 31, no. 4, pp. 617–636, 2014.

[7] J. Bhatti and T. E. Humphreys, "Hostile control of ships via false gps signals: Demonstration and detection," *NAVIGATION: Journal of the Institute of Navigation*, vol. 64, no. 1, pp. 51–66, 2017.

[8] A. Ranganathan, H. Ólafsdóttir, and S. Capkun, "Spree: A spoofing resistant gps receiver," in *Proceedings of the 22nd Annual International Conference on Mobile Computing and Networking*, 2016, pp. 348–360.

[9] E. U. 2021, "European gnss (galileo) open service signal-in-space interface control document issue 2.0 (galileo-os-sis-icd-2.0)," *European Union*, vol. 1, 2021.

[10] ——, "Galileo open service navigation message authentication (osnma) user icd for the test phase," *European Union*, vol. 1, 2021.

[11] ——, "Galileo open service navigation message authentication (osnma) receiver guidelines for the test phase," *European Union*, vol. 1, 2021.

[12] A. Perrig, D. Song, R. Canetti, J. Tygar, and B. Briscoe, "Timed efficient stream loss-tolerant authentication (tesla): Multicast source authentication transform introduction," 2005.

[13] J. Martin, J. Burbank, W. Kasch, and P. D. L. Mills, "Network Time Protocol Version 4: Protocol and Algorithms Specification," RFC 5905, Jun. 2010. [Online]. Available: https://www.rfc-editor.org/info/rfc5905

[14] C. Fernández–Prades, J. Arribas, P. Closas, C. Avilés, and L. Esteve, "GNSS-SDR: An open source tool for researchers and developers," in *Proc. 24th Intl. Tech. Meeting Sat. Div. Inst. Navig.*, Portland, Oregon, Sept. 2011, pp. 780–794.

[15] "galmon: galileo/GPS/GLONASS/BeiDou open source monitoring." [Online]. Available: https://github.com/berthubert/galmon

[16] D. Estévez, "galileo-osnma: Rust implementation of the galileo osnma (open service navigation message authentication) protocol," 2022. [Online]. Available: https://github.com/daniestevez/galileo-osnma

[17] "PyCryptodome." [Online]. Available: https://github.com/Legrandin/pycryptodome/

[18] M. Motallebighomi, H. Sathaye, M. Singh, and A. Ranganathan, "Cryptography is not enough: Relay attacks on authenticated gnss signals," 2022. [Online]. Available: https://arxiv.org/abs/2204.11641

[19] C. O'Driscoll and I. Fernandez-Hernandez, "Mapping bit to symbol unpredictability in convolutionally encoded messages with checksums, with application to galileo osnma," in *Proceedings of the 33rd International Technical Meeting of the Satellite Division of The Institute of Navigation (ION GNSS+ 2020)*, 09 2020, pp. 3751–3765.

[20] J. Clulow, G. P. Hancke, M. G. Kuhn, and T. Moore, "So near and yet so far: Distance-bounding attacks in wireless networks," in *Proceedings of the Third European Conference on Security and Privacy in Ad-Hoc and Sensor Networks*, ser. ESAS'06. Berlin, Heidelberg: Springer-Verlag, 2006, p. 83–97. [Online]. Available: https://doi.org/10.1007/11964254_9

[21] M. Flury, M. Poturalski, P. Papadimitratos, J.-P. Hubaux, and J.-Y. Le Boudec, "Effectiveness of distance-decreasing attacks against impulse radio ranging," in *Proceedings of the Third ACM Conference on Wireless Network Security*, ser. WiSec '10. New York, NY, USA: Association for Computing Machinery, 2010, p. 117–128. [Online]. Available: https://doi.org/10.1145/1741866.1741887

[22] C. Fernández–Prades, J. Arribas, P. Closas, C. Avilés, and L. Esteve, "GNSS-SDR general block diagram," 2013, [Online; accessed September 26, 2022]. [Online]. Available: https://github.com/gnss-sdr/gnss-sdr/blob/main/docs/doxygen/images/GeneralBlockDiagram.png

[23] J. Ametller and astromarc, "osnmapython: Osnma for galileo mass-market gnss receivers," 2021. [Online]. Available: https://github.com/astromarc/osnmaPython

[24] A. Galan, I. Fernandez-Hernandez, L. Cucchi, and G. Seco-Granados, "Osnmalib: An open python library for galileo osnma," in *2022 10th Workshop on Satellite Navigation Technology (NAVITEC)*, 2022, pp. 1–12.