# The security of GNSS systems

Levente Kovács[*]

2020-2021 spring semester

[*]EFOP-3.6.3-VEKOP-16-2017-00001: Talent Management in Autonomous Vehicle Control Technologies – The Project is supported by the Hungarian Government and co-financed by the European Social Fund.

# Contents

## 1 Introduction

A Global Navigation Satellite System (GNSS) refers to a satellite group in space that transmits positioning and time data to the receivers on a global scale. The receivers of this data can use it to determine their location. For most of its' history the GNSS market was dominated by the USA's Global Positioning System (GPS), but other nations have started to develop their own positioning systems, such as Galileo (maintained by the European Union and European Space Agency), Glonass (maintained by Russia) or Beidou (by China). The main focus of this paper is Europe's version of a GNSS, Galileo because of it's innovative, system-level security design. The other appealing characteristics of the system are that it's an independent navigation satellite system and has been tailored towards civil applications.

The biggest advantage of Galileo is that no governmental body can (officially) influence the availability of the system, unlike GPS, Glonass, and Beidou, which are all operated by their respective nation's department of defense and/or space agency. The availability of these systems can change to the general public if the operator decides to worsen the availability or reliability of the signal or outright block access for civil users (either due to some national conflict or other reasons). Galileo provides better accuracy than GPS, achieving 1 meter accuracy without corrections and 1.6 centimeters by using real-time corrections, GPS has 3 meters and 2.3 centimeters accuracy with these parameters respectively [1]. My main interest the application of the Open Service Navigation Message Authentication (OSNMA) security layer which can be used by any civil application to authenticate Galileo navigation data (or data sent by other GNSS). OSNMA is the first publicly available cryptographic security layer for GNSSs.

Since GPS is the most widely used GNSS it has the most attacks against applications using GPS for the user's location data. GPS servers can't authenticate the received data, so malicious entities can fabricate the GPS packets. To protect against such attacks, all receivers have to individually develop their anti-spoofing mechanisms. The cryptographic methods for this are only available for government-authorized personnel. Civil users can either use multi-constellation receivers, checking multiple GNSSs to cross-reference the received navigation data from all systems. An additional measure of protection is Inertial Measurement Units (IMU) which measures and reports a body's specific force, angular rate, and sometimes orientation. In theory, an adversary cannot spoof the Earth's gravitational field or vehicle dynamics and cause the IMU to think it has moved in a way that the movement is spoofed. These methods are unfortunately rather expensive, especially the latter.

This makes OSNMA quite attractive to use for civil applications, where these methods are not feasible to implement. Certain civil GPS applications are quite infamous for their hacker community, like Pokemon Go. There are available hacks to teleport around the map or move around in real-time, without actually moving, this is all controlled by the user. For Pokemon Go, there's no real-life consequence when someone hacks the system, but in situations where the location data is used to navigate vehicles (air-navigation, transporting goods, self-driving cars, etc.) with multiple passengers or expensive goods onboard, the spoofed location can cause accidents with much more severe outcomes.

Most type of applications in the recent decades have standardized the security protocols and applications to protect data-confidentiality, -integrity. The web has HTTPS, SSH can be used to secure shell connections, TLS can be used to protect any type of communications over the Internet. In the world of GNSS this sort of standardization didn't happen yet, there are no widely available protocols, but OSNMA can hopefully help pushing this agenda forward.

The contribution of this paper is starting the implementation of an OSNMA capable receiver. The implementation covers the cryptographic operations which are needed to process OSNMA and some of the data-parsing. Due to scarce testing data, I was unable to test some of these methods. I found some negative result during this work, finding some bugs and limitations in the Galileo and OSNMA systems which I discuss in this paper. Additionally, I have performed security analysis of the OSNMA system, reviewing some of the attacks known against GNSSs and checking how OSNMA would hold up against these attacks.

The structure of this paper is the following (excluding introduction and conclusion):

- General structure of OSNMA: Section 2 is about the OSNMA security layer's message structure and the necessary parameters.
- Cryptographic operations in OSNMA: Section 3 is about the cryptographic operations which are necessary to process OSNMA
- Availability of Galileo: Section 4 is about my findings on Galileo's availability and how the system is accessible for everyday users.
- OSNMA-capable receiver: Section 5 is about my work and results regarding the OSNMA-capable receiver's implementation.
- Threats on GNSS: Section 6 is about my analysis of commonly performed GNSS attacks and the protection of OSNMA against such threat agents.

## 2 The Open Service Navigation Message Authentication (OSNMA) security layer

Galileo's OSNMA provides a system-wide solution for navigation data authentication using cryptographic methods. It was implemented in the E1 band of Galileo, this signal contains navigation data, and paired with the OSNMA part of the message the received navigation messages can be verified and authenticated. The cryptographic methods are based on the TESLA (Timed Efficient Stream Loss-Tolerant Authentication) protocol [2].

The navigation messages are sent as pages, a full Galileo frame contains 24 sub-frames and each sub-frame contains 30 pages of data, with 15 odd and even pages. To assemble the sub-frame all of the 30 pages are needed, but the OSNMA data is only present in the odd pages, with 40 bits of data in each page. The 40 OSNMA bits can be further divided into 8 bits of HKROOT (Headers and Key Root) bits and 32 bits MACK (MAC & Key). The pages are sent in pairs (one odd, one even page) every 2 seconds, so each sub-frame is transmitted over a 30 second time window. A full-frame with 24 sub-frames is sent over a 720 second period.

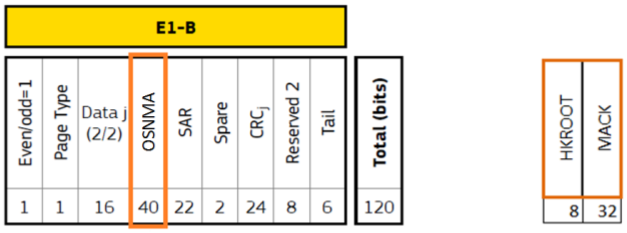


Figure 1: Structure of an odd E1-B page with the contained OSNMA data

### 2.1 HKROOT section

The HKROOT section of the OSNMA layer contains 120 bits of data per sub-frames with 8 bits per odd pages. The section contains 3 fields: NMA Header (8 bits), DSM Header (8 bits), and the DSM block (104 bits). The message signed in the DSM is usually the TESLA root key, but it can also be a new OSNMA public key.

The data which is sent to be authenticated by OSNMA are provided in blocks. In each subframe, the HKROOT parts provide a 104-bit Digital Signature Message (DSM) block. The full length of a DSM varies, but at most it contains 16 blocks (with 1664 bits of non-header data). The size depends on the cryptographic parameters and the message's type. Two types of DSM messages are possible: DSM-KROOT (DSM Key Root) and DSM-PKR (DSM Public Key Renewal). A DSM's first block always contains the number of blocks in

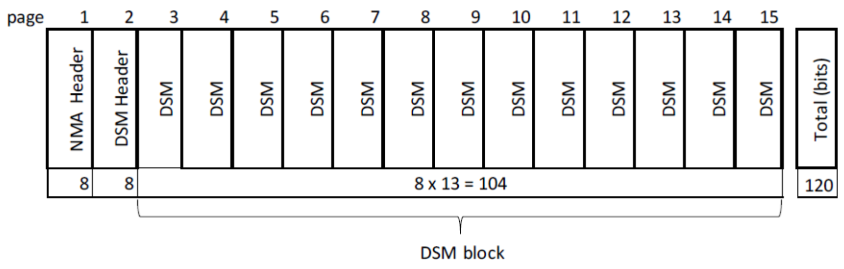


Figure 2: Structure of a DSM block

the current message. Using this the receiver has to store and order the received DSM blocks so it can assemble the full message.

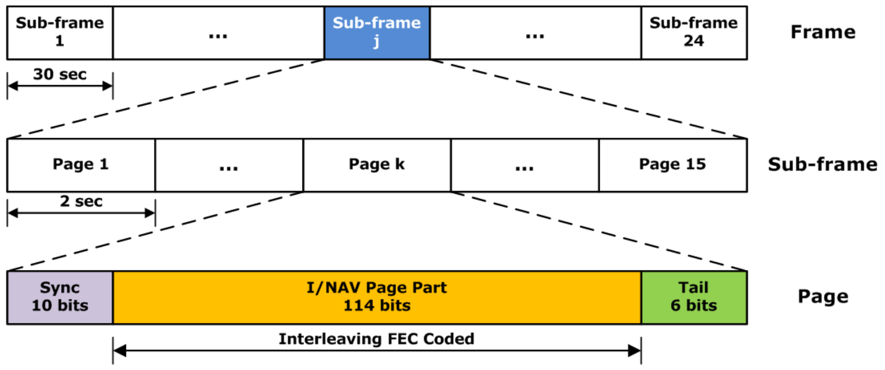


Figure 3: Structure of Galileo frames

The NMA and DSM headers provide information about the data contained in the DSM block. The NMA Header defines the status of the NMA service, with the following fields present:

- NMA Status: This field represents the status of the NMA service, with 4 values possible. The values are described as:
  - 0 - N/A: This value is not applicable.
  - 1 - Test: The NMA service is provided without any operational guarantees. Receivers may use and authenticate with the OSNMA data at their own risks.
  - 2 - Operational: The NMA service is provided as defined in the specification.
  - 3 - Don't Use: The receiver must not navigate using OSNMA data.

| NMA Status | Chain ID | Chain & PK status | Reserved | Total (bits) |
|---|---|---|---|---|
| 2 | 2 | 3 | 1 | 8 |

Figure 4: The NMA Header structure

- Chain ID: Represents the ID of the key chain in force. Supported values are 0 to 3, after which the CIDs overflow in a cyclic manner.
- Chain and Public Key Status: This field represents the status of the key chain, according to the following values:

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| Reserved | Nominal | EOC | CREV | NPK | PKREV | Reserved | Reserved |

Figure 5: Chain and Public Key Status values

  - 0 - Reserved: These values are reserved (for future use).
  - 1 - Nominal: The status of the chain, identified by the CID, and the public key in force, identified by PKID in DSM-KROOT, is nominal.
  - 2 - EOC (End Of Chain): The current chain is coming to an end. DSM-KROOT message will be transmitted regularly with the root key of the new chain.
  - 3 - CREV (Chain Revoked): A chain is or has been revoked. If NMA Status is "Don't Use", the current chain (associated with the transmitted CID) is revoked. If NMA Status is "Operational", then the chain associated with the previous CID has been revoked.
  - 4 - NPK (New Public Key): The public key is being renewed, this means a DSM-PKR message will be transmitted.
  - 5 - PKREV (Public Key Revoked): A public key is or has been revoked, depending on the NMA status: If NMA Status is "Don't Use", then the current public key, identified by the PKID in the DSM-KROOT, is revoked. If NMA Status is "Operational", then a

public key used in the past has been revoked (which is currently not in force).

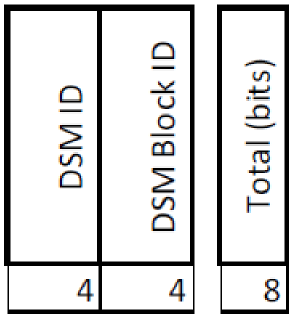
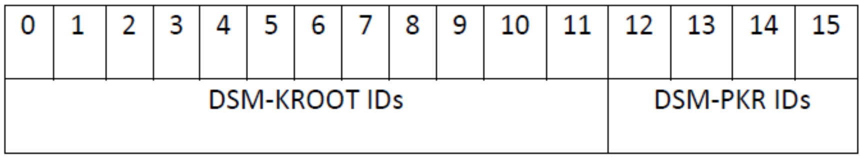


Figure 6: The DSM Header structure

The DSM header on the other hand provides information about the DSM and the block that's being transmitted. The fields are the following:

- DSM ID: It's a 4-bit identifier of the DSM. As the DSM is transmitted over several blocks, the DSM ID helps to identify which DSM's block was transmitted in the message. Also, this ID helps to identify the type of message received, DSM-KROOT or DSM-PKR. DSM IDs 0 to 11 correspond to DSM-KROOT messages, while 12 to 15 are reserved for DSM-PKR messages.



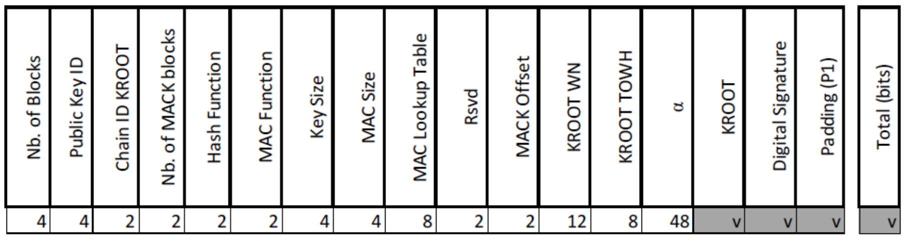
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| DSM-KROOT IDs | | | | | | | | | | | | DSM-PKR IDs | | | |

Figure 7: DSM ID values

- DSM Block ID: This field contains the ID of the DSM block transmitted in the current sub-frame. This identifies the block's position in the complete DSM, the first one is identified by BID = 0, second is BID = 1 all the way up to BID = 15, for a maximum of 16 blocks. In order to decode the full DSM message the receiver has to obtain all the DSM blocks are order them sequentially. The total number of blocks is described in the first block of the DSM (see DSM-KROOT and DSM-PKR).

## 2.2 DSM-KROOT

DSM-KROOT messages are responsible for providing and authenticating the TESLA root keys. It either provides authentication data to verify the root key in effect or a new key. The root key is verified by the ECDSA (Elliptic Curve Digital Signature Algorithm, as defined in the Digital Signature Standard [16]) with a public key provided by a DSM-PKR message or an OSNMA server [1]. Furthermore, it defines and verifies the cryptographic parameters needed to process the TESLA key-chain (Hash and MAC functions, key and MAC size, offset, etc.). Most of the time, DSM-KROOT messages are transmitted rather than DSM-PKR.

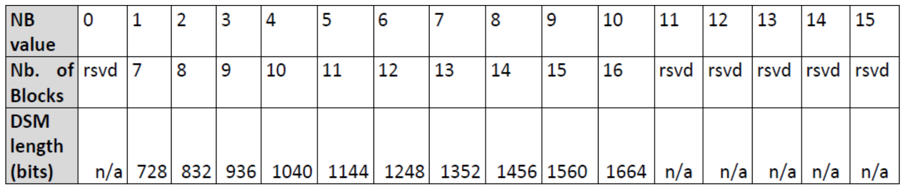


| Nb. of Blocks | Public Key ID | Chain ID KROOT | Nb. of MACK blocks | Hash Function | MAC Function | Key Size | MAC Size | MAC Lookup Table | Rsvd | MACK Offset | KROOT WN | KROOT TOWH | α | KROOT | Digital Signature | Padding (P1) | Total (bits) |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 4 | 4 | 2 | 2 | 2 | 2 | 4 | 4 | 8 | 2 | 2 | 12 | 8 | 48 | v | v | v | v |

Figure 8: Structure of DSM-KROOT messages

The size of the message depends on the length of the signature and the TESLA keychain. The parameters are the following:

- The number of blocks in the DSM. This 4-bit NB value doesn't directly contain the number of blocks, rather the Figure 9 contains the conversion table that can be used to determine the real block count.



| NB value | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Nb. of Blocks | rsvd | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | rsvd | rsvd | rsvd | rsvd | rsvd |
| DSM length (bits) | n/a | 728 | 832 | 936 | 1040 | 1144 | 1248 | 1352 | 1456 | 1560 | 1664 | n/a | n/a | n/a | n/a | n/a |

Figure 9: Number of blocks field with the associated total DSM length

- The public key used to create the signature (PKID). It's assumed that the receiver already possesses the key and algorithm associated with PKID, either because it is pre-installed on the system or was received by an OSNMA server.

[1]On the site https://www.gsc-europa.eu/ there will be a service which allows receivers to obtain the in-effect public-key in effect before starting operation, but this service is not available yet.

- The key-chain for which the root-key is provided (CIDKR)
- The number of MACK blocks in the sub-frame (NMACK). The number of MACK blocks can be either 1, 2 or 3, resulting in the size of a single MACK block being 480, 240 and 160 bits respectively in these settings.
- Hash function (HF) representing the hash function used for the key chain, 4 values are possible, 0 = SHA-256, 1 = SHA3-224, 2 = SHA3-256, 3 is reserved for future use. SHA-256 is compliant with the latest FIPS documentation on the SHA-2 family of hash functions [12]. SHA3 is implemented according to the Keccak algorithm [13].
- MAC function (MF) representing the MAC function which will be used with the key chain. It has 4 possible values, 0 = HMAC-SHA-256, 1 = CMAC-AES, 2 and 3 are reserved for future use. HMAC-SHA-256 is standardized in [14] and CMAC-AES is standardized as Algorithm 5 in [15].
- Key size (KS). The field itself can have 16 different values, with values over 8 being reserved for future use. The rest of the values correspond to the following key sizes: (KS=)0 - 96 bits, 1 - 104 bits, 2 - 112 bits, 3 - 120 bits, 4 - 128 bits, 5 - 160 bits, 6 - 192 bits, 7 - 224 bits, 8 - 256 bits.
- MAC size (MS). This field defines the degree of truncation for the MACs. It can have 16 different values, with values over 9 being reserved for future use. The used values correspond tot he following MAC sizes: (MS=)0 - 10 bits, 1 - 12 bits, 2 - 14 bits, 3 - 16 bits, 4 - 18 bits, 5 - 20 bits, 6 - 24 bits, 7 - 28 bits, 8 - 32 bits, 9 - 40 bits.
- The type of the authentication (MACLT). This 8-bit field defines the entry in a MAC lookup table which specifies the sequence of ADKD type (more on these later) values for the MACs in the MACK section.
- MACK offset (MO). If the value is 1, then some parts of the MACK blocks will be sent with a certain time delay (the delay is determined by MACLT)
- Parameters used for time-synchronization. KROOT WN - Week number of time associated with the KROOT, and KROOT TOWH - Time of week in hours which is associated with the KROOT.
- The $\alpha$ field contains a random bit sequence which is used during the hashing procedure.
- The KROOT field contains the public root key of the TESLA protocol.
- The DS field contains the digital signature produced by the ECDSA algorithm to verify some of the fields in the DSM-KROOT message.
- P1 field includes padding bits to be added to the DSM to fit a total length multiple of one DSM block.

An important property of OSNMA is that the MACs provided are truncated. It's generally not recommended in the world of cryptography to truncate a MAC by more than half its length, or no more than 80 bits with HMAC. However, GNSS authentication is very sensitive to signal conditions and the amount of data needed to transfer for authentication. OSNMA uses very small MAC tags, which are a result of this approach. Assuming the MAC algorithm is indistinguishable from a random lookup table (with key-message pairs) and random n-bit random sequences as values and given that authentication happens one-way, and the TESLA keys are only used once, a higher truncation of MACs seems to not increase the security risks significantly. It is very unlikely that an attacker can correctly guess even a few bits of the MACs when the time before authentication is small (10s of seconds). For example, the probability of guessing a 15-bit MAC correctly is $2^{-15} \approx 0.00003$, which is considered to be sufficiently low to discourage guessing attacks against the OSNMA system.

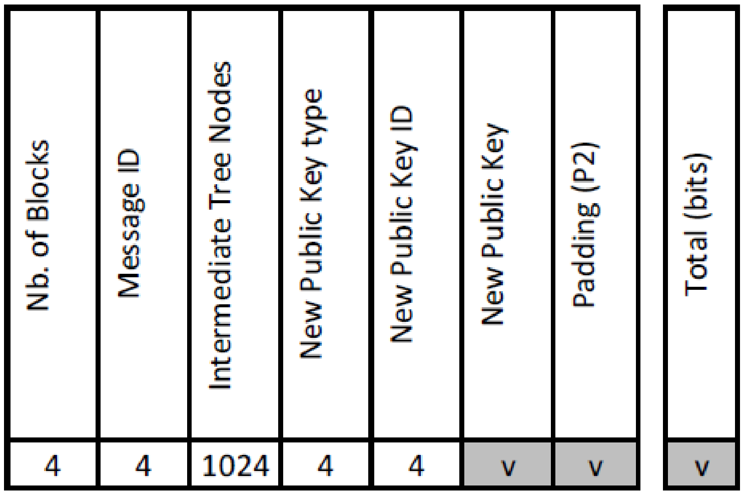
## 2.3 DSM-PKR

The DSM-PKR messages are used to renew the ECDSA public key used for the verification of the TESLA root key in the receiver. Additionally, DSM-PKR may contain a special EMS (Emergency Service Message), which when transmitted means, that the sender requests all receivers to stop using the authentication service and to connect to the OSNMA server for further details about continuing to use the service.

The received message (public key or EMS) can be verified by constructing a Merkle tree based on the received data. The leaves of the tree can be reconstructed based on some of the DSM-PKR's fields. The tree's root can be loaded in the receiver from the factory or retrieved from an OSNMA server. The SHA-256 hashing algorithm is used for the Merkle tree's reconstruction.

The size of the message depends on the size of the key, which is determined by the used ECDSA-variant. A DSM-PKR contains the following fields:

- NB: Number of blocks in the DSM-PKR message (same as with DSM-KROOT)
- MID: This field identifies which leaf of the Merkle tree is provided and the ITN's field's content. See figure 16.
- ITN: This field contains the necessary intermediate nodes for the authentication of the received leaf. The nodes are 256 bits long, needing 4 nodes to build the Merkle-tree, resulting in a total field size of 1024 bits.
- NPKT: Represents the new public key's associated signature algorithm. See figure 11.
- NPKID: Represents the ID of the received public key, if NPKT = 4, then NPKID is set to 0.



| Nb. of Blocks | Message ID | Intermediate Tree Nodes | New Public Key type | New Public Key ID | New Public Key | Padding (P2) | Total (bits) |
|---|---|---|---|---|---|---|---|
| 4 | 4 | 1024 | 4 | 4 | v | v | v |

Figure 10: The structure of the DSM-PKR messages

| PK type value | 0 | 1 | 2 | 3 | 4 | 5-16 |
|---|---|---|---|---|---|---|
| Type of message | ECDSA P-224 | ECDSA P-256 | ECDSA P-384 | ECDSA P-521 | Emergency Service Message | rsvd |

Figure 11: ECDSA-variants used with the associated NPKT value

- NPK: This field contains the new OSNMA public key, length will depend on the public key's type. For ECDSA P-224, ECDSA P-256, ECDSA P-384, and ECDSA P-521 length of the NPK will be 232, 264, 392, and 536 bits respectively.

  In case an emergency message is provided this field contains a random sequence of bits to allow the authentication of the message. The exact length of the NPK will be such that the whole DSM-PKR message fits in the number of blocks indicated by the NB field:

  $$L = (NB \cdot 104) - 1040$$

  Where $L$ is the length of the NPK field when an emergency message is provided, NB is the number of blocks field, and 1040 is the size in bits of all the other fields (NB, MID, ITN, NPKT, and NPKID)

- P2: This field includes padding bits added to the DSM if needed, such that the message fits into a total length of DSM blocks.

## 2.4 MACK section

The MACK part of the OSNMA layer is transmitted in parallel to the HK-ROOT. This section contains the MAC tags and time-delayed TESLA keys for the authentication of navigation data. Each odd page contains 32 bits of MACK data, adding up to 480 bits per sub-frame. A MACK block contains several MACs, each followed by a MAC-info field, providing information about the data authenticated, and a TESLA key. A sub-frame may contain multiple MACK blocks, the number of blocks depends on the TESLA key-chain parameters.

Every MACK's first block contains a special MAC0 tag, this authenticates important navigation data from the transmitting satellite. A MACSEQ tag is also present in the first block, which is used to authenticate the sequence of "flexible" MAC types that are transmitted in the current MACK block. Other MAC tags can authenticate other data provided by the satellite, the usage of these tags are determined by the ADKD (Authenticated Data & Key Delay) field in each of these MACK blocks.

The number of MACs per MACK block is the maximal possible, which fits into the block, and can be calculated as:

$$n_M = floor\left(\frac{480/NMACK - KS}{MS + 16}\right)$$

Where $n_M$ is the number of MACs, $NMACK$ is the number of MACK blocks, $KS$ is the key size, $MS$ is the MAC size. If the size of MAC & MAC-info sections plus the key size does not equal 160, 240, or 480-bits in length, then the spare bits are set to '0' at the end of each MACK block.

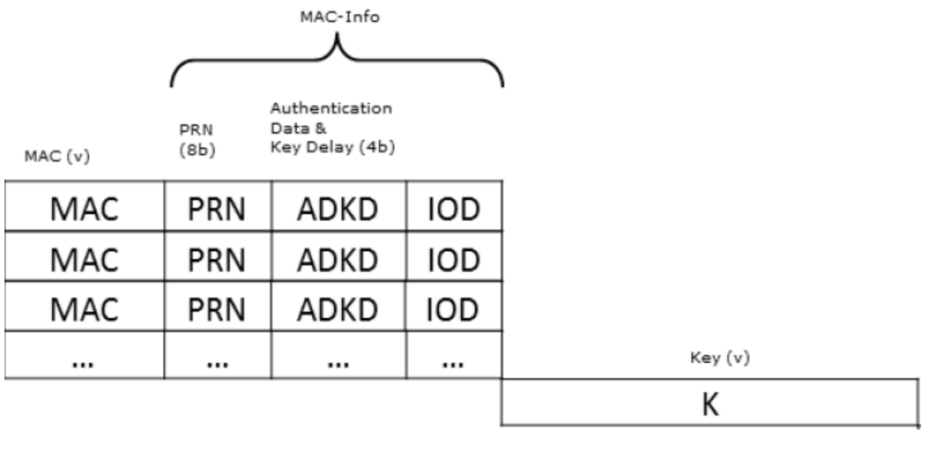
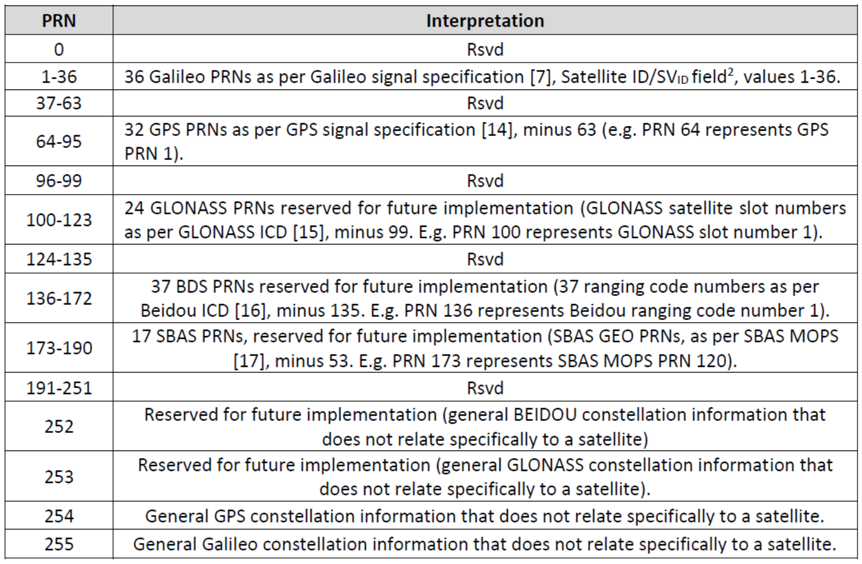


Figure 12: The structure of a MAC-section

The fields which are contained in the MACK blocks are the following, the first block is special, containing MAC0, MACSEQ and IOD fields only:

- MAC0, only present in the first MACK block, authenticates navigation data, more specifically: the first five words in the current sub-frame of the transmitting satellite. (ADKD $= 0$). The first five words of the sub-frame contain important navigation data (ephemeris, ionosphere, health check). The size of this section is determined by the MS field's value, transmitted in the DSM-KROOT message.
- MACSEQ, only present in the first MACK block, a 12-bit field that allows for the authentication of the MAC-info fields and MACs with "flexible" authentication type (FLEX type in ADKD)
- MAC, containing a MAC-tag, which authenticates the data determined by the ADKD field in the current section
- MAC-info, contains the PRN (pseudorandom noise, usable to authenticate transmitting satellites), ADKD, and IOD fields. It identifies the data which needs to be authenticated by using the MAC field.
- PRN contains the pseudorandom noise of the transmitting satellite for which the data needs to be authenticated. It uses 8 bits of data and this allows the authentication of 255 satellites' data, from Galileo and some other constellations. The implementation of the cross-authentication mechanism has started for the GPS, but other systems are also planned to be implemented in the future.



| PRN | Interpretation |
|---|---|
| 0 | Rsvd |
| 1-36 | 36 Galileo PRNs as per Galileo signal specification [7], Satellite ID/$SV_{ID}$ field[2], values 1-36. |
| 37-63 | Rsvd |
| 64-95 | 32 GPS PRNs as per GPS signal specification [14], minus 63 (e.g. PRN 64 represents GPS PRN 1). |
| 96-99 | Rsvd |
| 100-123 | 24 GLONASS PRNs reserved for future implementation (GLONASS satellite slot numbers as per GLONASS ICD [15], minus 99. E.g. PRN 100 represents GLONASS slot number 1). |
| 124-135 | Rsvd |
| 136-172 | 37 BDS PRNs reserved for future implementation (37 ranging code numbers as per Beidou ICD [16], minus 135. E.g. PRN 136 represents Beidou ranging code number 1). |
| 173-190 | 17 SBAS PRNs, reserved for future implementation (SBAS GEO PRNs, as per SBAS MOPS [17], minus 53. E.g. PRN 173 represents SBAS MOPS PRN 120). |
| 191-251 | Rsvd |
| 252 | Reserved for future implementation (general BEIDOU constellation information that does not relate specifically to a satellite) |
| 253 | Reserved for future implementation (general GLONASS constellation information that does not relate specifically to a satellite). |
| 254 | General GPS constellation information that does not relate specifically to a satellite. |
| 255 | General Galileo constellation information that does not relate specifically to a satellite. |

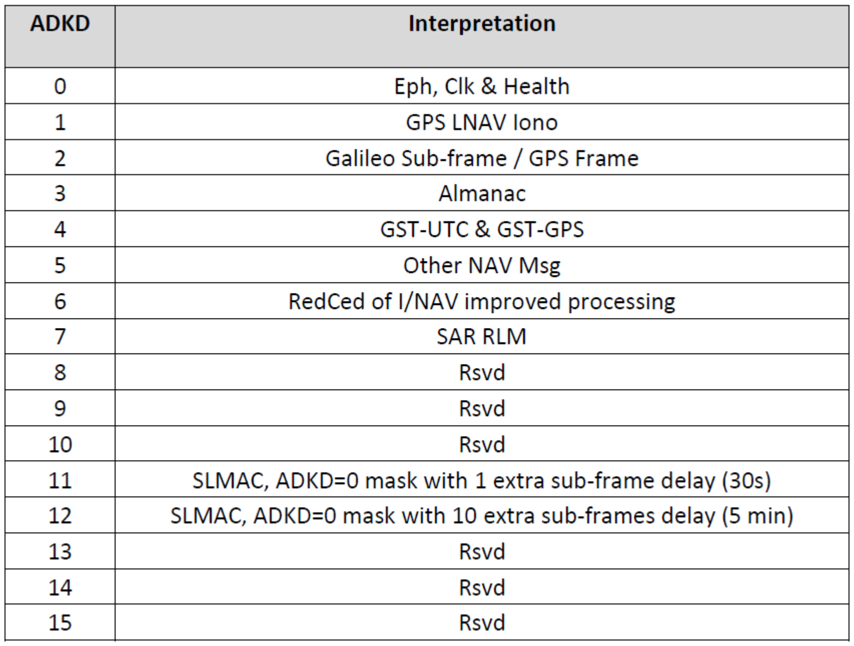
Figure 13: The definition of the PRN-section

A special case is when ADKD $= 7$ (SAR RLM authentication), then the

PRN field corresponds to the 8 bits of the MSB-truncated 24-bit CRC calculated over the concatenation of the SAR RLM identifier bit and the 60-bit beacon ID field as follows:

$$SARPRN = trunc(8, CRC(SAR\ RLM\ id || Beacon\ id)$$

- ADKD (Authentication Data & Key Delay) describes the authenticated navigation data, and in cases of ”slow MAC” values, the extra time needed for the key to be disclosed, after the MAC was already sent.



| ADKD | Interpretation |
|---|---|
| 0 | Eph, Clk & Health |
| 1 | GPS LNAV Iono |
| 2 | Galileo Sub-frame / GPS Frame |
| 3 | Almanac |
| 4 | GST-UTC & GST-GPS |
| 5 | Other NAV Msg |
| 6 | RedCed of I/NAV improved processing |
| 7 | SAR RLM |
| 8 | Rsvd |
| 9 | Rsvd |
| 10 | Rsvd |
| 11 | SLMAC, ADKD=0 mask with 1 extra sub-frame delay (30s) |
| 12 | SLMAC, ADKD=0 mask with 10 extra sub-frames delay (5 min) |
| 13 | Rsvd |
| 14 | Rsvd |
| 15 | Rsvd |

Figure 14: General meaning of ADKD values

- IOD (Issue of Data), identification of the issue of data for the authenticated information. It needs to be interpreted based on the ADKD.
- Key, this field contains a key from the TESLA key-chain, the position in the chain depends on the satellite PRN and time (more on that later).

## 2.5 Cross-Authentication

GNSS constellations other than Galileo are planned to be available in the future to authenticate with OSNMA. Only high-level test plans are available for this functionality at this point, but the work has been started for the GPS. This cross-authentication service is different than what’s defined in the NMA setting.

GNSS authentication data is usually delivered on a per-satellite basis, which leads to the following constraints:

- Possibly several hundreds of bits are needed to be decoded from each operating satellite for a single authentication procedure (of all satellites). This is problematic because satellites with bad visibility may transmit the authentication data with an unacceptable bit-error-rate
- Each satellite has to be able to transmit its authentication data
- Satellites which are not providing NMA can't have their data authenticated by other satellites (satellites from other GNSS can't use the system and are therefore excluded)

To circumvent these issues a satellite could transmit authentication information for other surrounding satellites. In this setting, each satellite is sending its normal, plaintext navigation data as usual, but there are designated, special satellites sending authentication data not only for their own authentication but other satellites as well.

The power of this approach is that the "normal" satellites, ones that are only sending navigation data, can be part of other GNSS rather than using Galileo. In theory, this approach could allow the authentication of any GNSS satellite using this approach. However, cross-authentication has a key difference from the standard authentication methods in terms of functionality. In standard authentication methods, there's only one source that is authenticated, the source providing both the plaintext and authentication data. In cross-authentication, the sender of the plaintext and authentication data are different sources. See figure 15 for an illustration on the idea of cross-authentication.

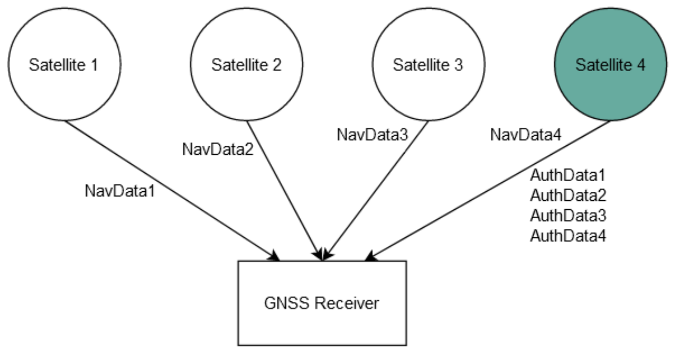


Figure 15: Cross-authentication design illustration

It's also possible to optimize this approach (with regards to the navigation and authentication bits required for transmission between satellites and receivers) by combining AuthData 1-3 with AuthData 4. The sending satellite has access to the plaintext data, so it can concatenate its own NavData 4 with the other NavData. This way, AuthData4 becomes obsolete, and AuthData1-4, AuthData2-4, and AuthData 3-4 could be sent instead. The joined authentication tag can be verified by using the NavData of both satellites.

Having all satellites supporting cross-authentication provides authentication for all visible satellites to the receiver by those with better visibility conditions.

As mentioned before, it's been stated that OSNMA is planning to support other GNSS constellations with cross-authentication. This includes GPS, GLONASS, Beidou and SBAS (Satellite-Based Augmentation System) satellites [7]. Using the general ADKD interpretation one can assume how an implementation on another constellation would look like, the defined parameters are part of every GNSS, but unfortunately, there's only a vague idea about how cross-authentication will look like on the data level for these systems. There needs to be more research done regarding the other systems to collect the necessary parts from the navigation data, which are then authenticated corresponding with the ADKDs.

There's a concern with high availability, to be able to authenticate plaintext satellites an authenticating satellite also has to receive the plaintext data and then forward the authentication data for it. Currently, OSNMA is planned to have 26 operating satellites and the maximum number of MACK bits that a satellite sends per sub-frame is 480 (may contain multiple MACK blocks, see later). This results in a total throughput of $26 \cdot 480 = 12480$ bits per sub-frames. In these 480 bits, a satellite can send around 5-15 MACs, but this number heavily depends on the cryptographic parameters used. It's unclear how many MACs are needed in a MACK block per plaintext satellite to cross-authenticate its data (this is up to future implementation), but at least 2-3 is needed to authenticate the necessary parts (Eph, Clk & Health, Almanac, etc.). This opens up the question of whether the cross-authentication will require extra authenticating satellites or some of the weaker cryptographic parameters supported by OSNMA to be functional.

Work with GPS has already been started, the navigation data supplied by GPS is mostly the same as with Galileo: ephemeris, ionosphere, health, and almanacs. The first three are transmitted every 30-seconds in the GPS L1 C/A code. This allows OSNMA to authenticate a new set of ephemeris data for GPS every 30 seconds. To authenticate almanacs, the requirement is the reception of a full 12.5 minute GPS frame. If a new almanac is obtained, the OSNMA MAC can be generated and sent by the authenticating satellite.

# 3 Cryptographic operations in an OSNMA capable Galileo receiver

This section describes the cryptographic operations needed to support and use OSNMA data for authentication in the Galileo system. These operations can be divided into 4 categories:

1. DSM-KROOT operations: reception and verification of the root key for a TESLA key-chain
2. DSM-PKR operations: reception and verification of a new public key
3. TESLA key operations: reception and verification of TESLA keys with a root-key obtained from DSM-KROOT
4. MAC operations: verification of navigation messages

## 3.1 DSM-KROOT

The DSM-KROOT contains the in-effect, or nextly usable TESLA root key and the data needed for the verification and interpretation of this message. The receiver has to verify the root-key using the ECDSA signature, supplied with the message. The digital signature is created using the following data fields:

$M$ = (NMA-Header || CIDKR || NMACK || HF || MF || KS || MS || MACLT || Rsvd || MO || KROOT WN || KROOT TOWH || $\alpha$ || KROOT)

Where NMA-Header is transmitted in parallel to the DSM-KROOT message in every DSM block (containing info about the key-chain). The rest of the fields are the same values that are present in the DSM-KROOT message. It's needed to concatenate the $P0$ padding to $M$ as well, the message is then signed with a $k$ key as the following:

$DS = sign(k, M||P0)$

The public key used for the verification of the ECDSA signature has to be obtained by the receiver either by checking out the key from the OSNMA server or it was renewed by a DSM-PKR message.

A $P1$ padding field is added to the DSM-KROOT message as follows:

$$P1 = trunc(L, hash(M||P0))$$

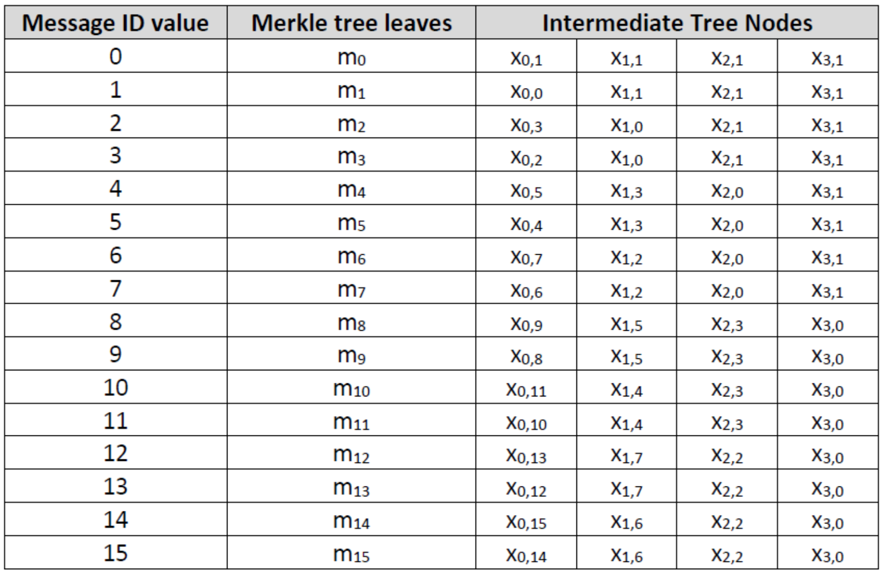
Where $L = 104 \cdot NB - length(M) - length(DS)$.

The $P1$ padding bits are added to the DSM and $L$ is the difference in the number of bits between the total block length and the length of the $DS$ and $M$. The $hash$ used is SHA-256. The ECDSA variant has to be known when taking the new public key into effect, which can be read from the NPKT field of the DSM-KROOT message.

## 3.2 DSM-PKR

A DSM-PKR message contains a new ECDSA public key and its' cryptographic parameters and the data needed to authenticate the DSM. The verification has to be done by rebuilding a Merkle-tree out of the received data. The root of the tree has to be obtained by the receiver before starting to use the OSNMA service. The DSM-PKR fields can be used to reconstruct one of the tree leaves and using the intermediate nodes the Merkle-tree can be rebuilt locally in the receiver. The $l$ leaf is reconstructed as the following:

$$l = (\text{NPKT} \mid\mid \text{NPKID} \mid\mid \text{NPK})$$



| Message ID value | Merkle tree leaves | Intermediate Tree Nodes | | | |
|---|---|---|---|---|---|
| 0 | $m_0$ | $x_{0,1}$ | $x_{1,1}$ | $x_{2,1}$ | $x_{3,1}$ |
| 1 | $m_1$ | $x_{0,0}$ | $x_{1,1}$ | $x_{2,1}$ | $x_{3,1}$ |
| 2 | $m_2$ | $x_{0,3}$ | $x_{1,0}$ | $x_{2,1}$ | $x_{3,1}$ |
| 3 | $m_3$ | $x_{0,2}$ | $x_{1,0}$ | $x_{2,1}$ | $x_{3,1}$ |
| 4 | $m_4$ | $x_{0,5}$ | $x_{1,3}$ | $x_{2,0}$ | $x_{3,1}$ |
| 5 | $m_5$ | $x_{0,4}$ | $x_{1,3}$ | $x_{2,0}$ | $x_{3,1}$ |
| 6 | $m_6$ | $x_{0,7}$ | $x_{1,2}$ | $x_{2,0}$ | $x_{3,1}$ |
| 7 | $m_7$ | $x_{0,6}$ | $x_{1,2}$ | $x_{2,0}$ | $x_{3,1}$ |
| 8 | $m_8$ | $x_{0,9}$ | $x_{1,5}$ | $x_{2,3}$ | $x_{3,0}$ |
| 9 | $m_9$ | $x_{0,8}$ | $x_{1,5}$ | $x_{2,3}$ | $x_{3,0}$ |
| 10 | $m_{10}$ | $x_{0,11}$ | $x_{1,4}$ | $x_{2,3}$ | $x_{3,0}$ |
| 11 | $m_{11}$ | $x_{0,10}$ | $x_{1,4}$ | $x_{2,3}$ | $x_{3,0}$ |
| 12 | $m_{12}$ | $x_{0,13}$ | $x_{1,7}$ | $x_{2,2}$ | $x_{3,0}$ |
| 13 | $m_{13}$ | $x_{0,12}$ | $x_{1,7}$ | $x_{2,2}$ | $x_{3,0}$ |
| 14 | $m_{14}$ | $x_{0,15}$ | $x_{1,6}$ | $x_{2,2}$ | $x_{3,0}$ |
| 15 | $m_{15}$ | $x_{0,14}$ | $x_{1,6}$ | $x_{2,2}$ | $x_{3,0}$ |

Figure 16: Message ID field and associated Merkle tree leaves and intermediate tree nodes

After the $l$ leaf is reconstructed from the received DSM-PKR fields, rebuilding the tree can be started. The depth of the Merkle-tree is 4, allowing for the authentication of 16 public keys and emergency service messages. The intermediate nodes of the tree are received in the ITN field, the contents of this field depend on which leaf of the tree was transmitted (see Figure 16), based on the MID it can be determined how to process the received data.

The tree is reconstructed by concatenating the leaf and the intermediate nodes then hashing them to create the yet unknown intermediate nodes. More precisely, leaves have to be generated as the following:

$$\forall\ i = 0 \ldots 15,\ \ x_{0,i} = hash(m_i)$$

While the intermediate nodes can be determined as:

$$x_{j,i} = hash(x_{j-1,2\cdot i}||x_{j-1,2\cdot i+1}),\ j \in \{1,2,3,4\}$$

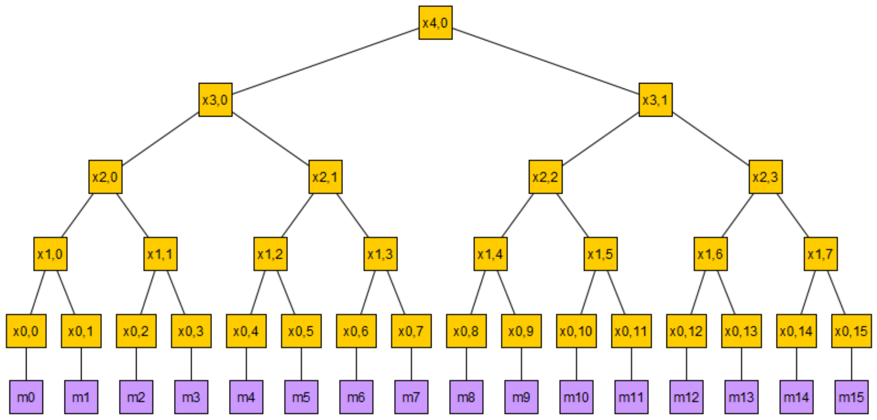


Figure 17: Labels of the OSNMA Merke-tree

To verify the DSM using the root of the tree, first, the leaf has to be regenerated, for example in case $i = 0 : x = hash(m_0)$. After, the received MID determines the processing, mainly, from which side (left or right) the intermediate nodes have to be concatenated to the generated nodes, in order to get the proper root at the end. In case $i = 0$ the intermediate nodes all have to concatenated from the right $x_{1,0} = hash(x_{0,0}||x_{0,1})$, etc. When the final hash is computed, it needs to be compared with the tree-root stored on the receiver's side, if the two values are equal, then the verification is successful.

## 3.3 Verification of TESLA keys

The TESLA key chain starts with a randomly generated (private) seed key, which is only known to the OSNMA provider. At the end of the key chain, there will be a $K_0$ root-key which is then publicized in Galileo sub-frames and authenticated by DSM-KROOT. More formally, the key-chain is generated by starting with the random $K_n$ seed key and the following $F$ one-way function, $n$ is the length of the key chain:

$$K_m = F(K_{m+1}) = trunc(k_{len}, hash(K_{m+1}||GST_{SF}||\alpha||P3))$$

Where $m \in \{0, 1, ..., n-1\}$, $k_{len}$ is the key length defined in the key chain parameters, $GST_{SF}$ is the Galileo System Time at the start of the 30-second interval when the sub-frame, in which the $K_m$ key will be transmitted, is being relayed. On the E1 band, this time can be obtained by subtracting 1 from the beginning timestamp of the sub-frame.

It's important to note that each key chain provides keys for multiple satellites, so each time TESLA keys are transmitted multiple keys are obtained. The keys which were transmitted in parallel form a block of keys, to decide which key was transmitted, the receiver has to check the satellite's PRN to determine the key's position in the chain.

The key generator has to provide $NS$·NMACK keys per sub-frame, where $NS$ is the number of satellites transmitting OSNMA and NMACK is the number of MACK blocks per sub-frame. With this construction: each MACK will have a unique key for every OSNMA provider satellite.

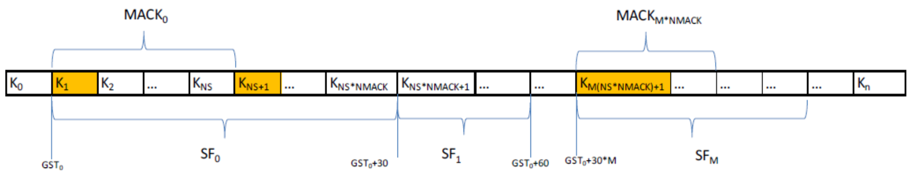


Figure 18: MACK keys as transmitted in a MACK block

Looking at figure 18, see that the $K_1, K_{NS+1}, K_{2 \cdot NS+1}, ...$ keys are always transmitted by the satellite with $PRN = 1$. The $K_2, K_{NS+2}, K_{2 \cdot NS+2}$ keys are transmitted by $PRN = 2$ etc. The keys in these MACK blocks are used to generate the MACs transmitted, except if the ADKD is a slow MAC type and the MACK offset is turned on, in which case the arrival of the keys will be delayed by a certain number of sub-frames ($1, 5$ or $10$, depending on the ADKD).

When a receiver obtains a $K_j, j \in \{1, ..., n-1\}$ key, the verification is done according to the TESLA protocol. The $K_0$ root key for the key-chain has to be obtained from a DSM-KROOT message before verifying the given key. To verify $K_j$, the $F$ function has to be applied $j$ times to $K_j$ and check if the result of this operation is the $K_0$ root key. More formally:

$$F^j(K_j) = K'_0$$

$$K'_0 =^? K_0$$

Where $F^j(K_j)$ is $F$ applied to $K_j$ $j$ number of times. If $K'_0 = K_0$, then $K_j$ is successfully verified. To avoid always repeatedly applying $F$ until getting $K_0$, the receiver can cache the $K_j$ key as the newest verified key. The next verification attempt shall be repeated until getting $K_j$, then the new key can be cached again for the next verification and so on.

## 3.4 MAC verification

The non-MAC0 tags sent in the OSNMA layer are generated in the following way:

$$m = (PRN_N||PRN_A||GST_{SF}||CTR||NMA_{status}||navdata||P3)$$

$$tag = trunc(MS, mac(k, m))$$

While in case of MAC0:

$$m_0 = (PRN_A||GST_{SF}||CTR||NMA_{status}||navdata||P3)$$
$$tag_0 = trunc(MS, sign(k, m_0))$$

Where the parameters are the following:

- $m, m_0$ are the authenticated messages
- $PRN_N$ is the $PRN$ (satellite ID) of the satellite which transmits the authenticated message
- $PRN_A$ is the $PRN$ of the satellite which provides the authentication (can be different than $PRN_N$ in case of cross-authentication)
- $GST_{SF}$ is the Galileo System Time in seconds at the start of the sub-frame in which the MAC is transmitted
- $CTR$ is the MAC's position in the MACK block in which it's transmitted
- $NMA_{status}$: This value is the same as in the NMA-Header's $NMA_{status}$ in the subframe's DSM block
- $navdata$ is the authenticated navigation data, collected from the sub-frame as specified by the ADKD
- $P3$ is the number of 0 padding bits (0-7) in the message to have a whole-byte length
- $tag, tag_0$ is the truncated MAC which is transmitted by the satellite to the receivers
- $MS$ is the size of the MAC
- $k$ is the key from the TESLA key-chain

To authenticate the satellite, the $tag$ has to be reconstructed locally in the receiver based on the received data. The $sign$ function is the signing of the appropriate MAC-type, defined in the DSM-KROOT message. To reconstruct the MAC, the data has to signed using the received key (after checking its validity with the TESLA root key) and the $sign$ function. If the reconstructed tag matches the received $tag$ from the satellite then the message is authenticated.

## 3.5 MACSEQ verification

The MACSEQ field contains a MAC, constructed from the MAC-info fields where the ADKD value is set to "flexible". The tag is constructed as follows:

$$m = (PRN_A||GST_{SF}||MFLEX_1||MFLEX_2||...||MFLEX_N)$$
$$MACSEQ = trunc(12, sign(k, m))$$

Where the parameters are defined as:

- $m$ is the authenticated message
- $PRN_A$ and $GST_{SF}$ are the same as with MAC verification
- $N$ is the number of MACs transmitted in the MACK block where the ADKD is set to "flexible"
- The $MFLEX_i$ fields are the MAC-info fields that need to be authenticated. $MFLEX_i$, $i \in \{1, .., N\}$, is $i$'th MAC with flexible ADKD in the MACK block. All the flexible MAC-infos are concatenated to receive the authenticated $m$ message
- $k$ is a key from the TESLA key-chain

The authentication of the $MACSEQ$ tag is done similarly to MACs in the previous chapter. The $m$ message has to be reconstructed from the data received and the $MACSEQ$ tag is reconstructed. If the resulting tag matches the received one the authentication is successful.

## 4 The availability of Galileo

Galileo isn't a traditional GNSS and it's an experimental technology, more and more devices support Galileo each day, but to be able to fully use the system's capabilities it's not enough to just capture Galileo. The full infrastructure is planning to have 24 satellites, but at the time of writing only 21 satellites are operating [3]. This may affect the availability of the service in some parts of the world.

The devices supporting Galileo are listed on the official UseGalileo website [4]. Unfortunately, not all devices support Galileo in an equal amount, while all of them are in the UseGalileo database, we found out that most smartphone devices are unable to use the OSNMA layer because it requires special navigation message tracking functionality to be able to access the Navigation Messages sent by the system. Some chipsets also have a problem when it comes to decoding the navigation messages, phones with Qualcomm chips (for example Samsung Galaxy A51) are unable to properly receive and interpret the navigation message. Devices with Broadcom chips (like Xiaomi MI 8, or Samsung S series) are successful at decoding the received messages. Unfortunately, there's a next problem: the tracking of navigation messages has to happen continuously on the E1 band of Galileo (its' the data component, where OSNMA is available). Tracking should be based on the Galileo data component for every GNSS measurement, this way every request's answer will contain the Galileo navigation messages containing the OSNMA.

The last available documentation about the OSNMA availability in smartphones mentions these issues and also that there is a firmware bug with the continuous data component tracking (stopping tracking of the data component after 5 minutes), but it's unknown whether this bug got fixed or not in a device released later. On the other hand, most phones actually use pilot tracking, which means that the GNSS chipset tracks the data component only at the start of a location request. When the navigation data is obtained the receiver switches to Pilot code (E1-C, instead of E1-B where the OSNMA is present), which means no navigation data is received. This happens, because the receiver has to synchronize with the satellite transmitted signal. Galileo uses the E1 Open Service modulation, multiplexing two components together:

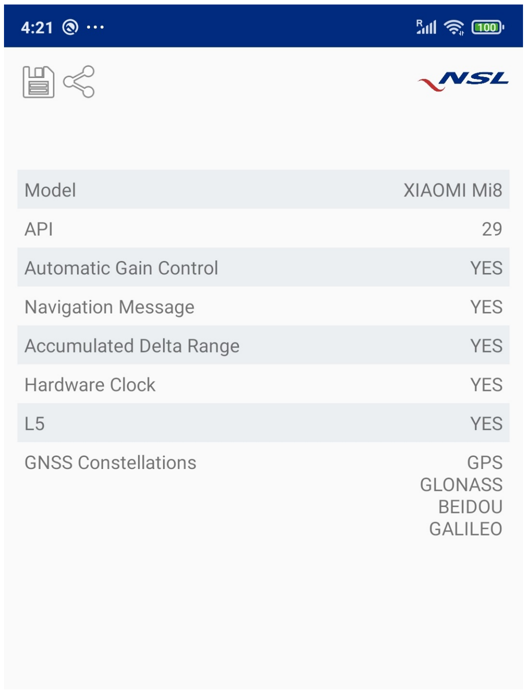
- E1B: The E1 data component, it is generated from the I/NAV navigation data stream and the primary code (E1BC).
- E1C: The E1 pilot component, composed of the primary code (E1BC) and secondary code (E1C_2nd).

Most smartphones only track the data component at the start of the request, because after a few exchanges the receiver is synchronized with the E1C_2nd code lock. Most receivers switch to tracking the pilot component when the ephemerides and the clock are obtained through an external source or by decoding a navigation message. After the synchronization happens with the satellite the receiver can compute the GNSS time and the chip will not continue to track the E1 data component. This is because pilot tracking allows the receivers to track the signal coherently for longer periods. The data component's absence allows the receiver to work with a much weaker signal with the GNSS as before, resulting in a more robust tracking mechanism.

A possible workaround to still capture some navigation messages on phones with pilot tracking is to reset the location request every few seconds, this ensures that more navigation pages arrive. This works on Android devices. I've developed a simple GNSS dumper application that uses this method to retrieve Galileo pages on a Xiaomi MI 8 device. These issues have also been reported by Airbus [5].

The GNSS Check application can be used to see whether the phone supports Galileo, navigation message tracking and the necessary API level (Android API level 24 is needed to use GNSS Navigation Messages) to use these GNSS functionalities [6].

The Android API allows users to capture the Galileo navigation pages by using the GnssNavigationMessage class. Most importantly, this supplies the byte-level dump of the navigation page data, but also some extra information about the contents of the message.



| | |
|---|---|
| Model | XIAOMI Mi8 |
| API | 29 |
| Automatic Gain Control | YES |
| Navigation Message | YES |
| Accumulated Delta Range | YES |
| Hardware Clock | YES |
| L5 | YES |
| GNSS Constellations | GPS GLONASS BEIDOU GALILEO |

Figure 19: GNSS Check diagnostics for Xiaomi Mi8 device

This doesn't include the pilot/data tracking functionality and unfortunately, there's no publicly available documentation about which devices support which functionalities listed as needed to use OSNMA. An open-source database about capable devices have been created, after Google removed their documentation with detailed device analysis about GNSS functionalities there was no widely available documentation for some time. The current state of the database about capable devices can be seen here [17].

In response to these hardware problems, the ESA replied that they are aware and are planning to actively work with chip manufacturers to tackle these issues. A big part of the authentication system relies on the OSNMA-capable receiver units, which is not possible without qualified hardware. Hopefully, new chipsets will start supporting Galileo OSNMA without major bugs in the near future and authentic navigation data will be available to a wider audience.

I captured data using 2 smartphones while I was working on this paper. First using Samsung Galaxy A51, which unfortunately wasn't able to record the Galileo data-pages properly. The format of the recorded data did not match the Android and Galileo documentations. As mentioned before, this was due to the Qualcomm chip not supporting the decoding procedure. Second, using the Xiaomi MI 8 smartphone I was able to record only a limited amount of OSNMA data (due to the pilot tracking functionality), but it was properly decoded and could be analyzed. It is unfortunately not usable however for testing without the necessary cryptographic materials.

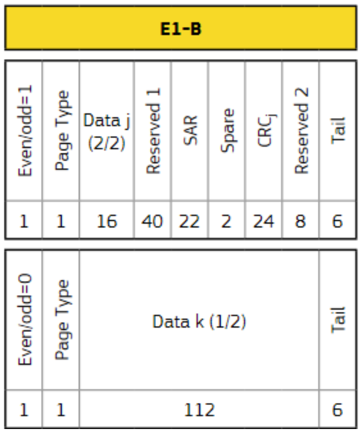
Unfortunately, the Galileo OSNMA signal's public testing is still underway, the first internal tests used 2 Galileo satellites on November 18th, 2020. This was the first-ever signal-in-space test with the authentication service, using reserved bits in the E1B signal. These internal tests are unfortunately not targeted toward the public and as such the gathering of live OSNMA data for testing is still not available in practice. The Galileo constellation has been transmitting the OSNMA test signal since November 18th for intermittent periods. It's been promised that this will continue over the next few months. After the internal testing has been completed successfully, the public observation phase will begin. In this stage, OSNMA will be publicly accessible and the European Space Agency will release some highly anticipated documentation about the OSNMA system (OSNMA Signal-in-Space Interface Control document, receiver implementation guidelines, and the necessary cryptographic materials). During the public testing phase, users will be able to send their feedback about the system and report any bugs that may occur [18].

# 5 Galileo OSNMA-capable Receiver Implementation

I have implemented the OSNMA cryptographic operations as a prototype as a form of Python scripts. Some of these operations needed supporting data-parsing methods, to be able to collect the necessary data for authentication from a Galileo I/NAV subframe. This is done by collecting the authenticated data from the received subframe's navigation data based on the ADKD field, parsing DSM-headers, etc. The scripts perform the necessary authentication methods as defined in this paper and the official OSNMA documentation [7]. The scripts use the test data supplied by the same document. The navigation page data is described in more detail in the Galileo Interface Control Document, defining the general Galileo data structures [8].

This implementation covers the cryptographic layer in OSNMA, but due to the delay of the OSNMA public-testing, test data is scarce. The data released by the ESA didn't contain test data for some of the ADKD authentication types and some cryptographic configuration options. Since no test data was available for these operations, the usage of these parameters is not implemented. In December 2020, the internal testing of the OSNMA layer has started. Capturing live data for the testing is now possible in some intermittent periods, but the necessary cryptographic materials remain unpublished, making most of the data we captured unprocessable. Most recently, the ESA promised that OSNMA will begin its public-observation period in the summer of 2021.

Data-capturing is a bit complicated, as it requires special functionalities. I've used a Xiaomi MI 8 smartphone to capture live Galileo data, by implementing a small GNSSDump application. This app implements the location request resetting, mentioned in the previous chapter, and logs the received GNSS navigation messages.


E1-B

| Even/odd=1 | Page Type | Data j (2/2) | Reserved 1 | SAR | Spare | CRC$_j$ | Reserved 2 | Tail |
|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 16 | 40 | 22 | 2 | 24 | 8 | 6 |

| Even/odd=0 | Page Type | Data k (1/2) | Tail |
|---|---|---|---|
| 1 | 1 | 112 | 6 |

Figure 20: E1-B page structure

The captured navigation message pages are part of the Galileo E1B signal. The receiver needs to be able to correctly assemble the Galileo sub-frame from these pages and then collect the OSNMA parts to create the DSM block. To be able to correctly order the navigation pages the receiver needs to parse the navigation data part of the message and identify the word type. Each pair of even and odd pages contains 128 bits of navigation data. These bits resemble the Galileo I/NAV words, the first 6 bits of each word identifies the type of the word.

The receiver has to know the Galileo sub-frame structure and correctly place the data blocks into the sub-frame based on the word type and the time of reception. A single word can be gathered from two sequential pages, when receiving a page, the first bit decides whether it's even or odd. If it's 0 the page is even, if 1 it's odd. The even pages contain the first 112 bits of a Galileo word and the next odd page contains the remaining 16 bits (odd pages also contain the OSNMA bits, see Reserved 1 on 20). The first 6 bits of this 128 block represent the type of Galileo word, the rest of the content depends on this type.

Using the live data, I've been able to verify that the Galileo signal contains the OSNMA bits and use this data to test the DSM-header parsing methods.

# 6 Threat model for GNSS

GNSS systems have seen numerous attacks throughout the years, however, the main types for attacks are still similar. For almost all attacks the goal is meaconing, intrusion, jamming, or interfering with the system. In general, disrupt the availability or performance of the navigation system. These attacks can be performed by malicious or untrusted entities targeting other devices (trusted or untrusted) to the system. Some disruption can happen unintentionally, but most attacks are performed with malicious intent in mind. Attacks can be performed on different levels of the GNSS ecosystem:

- Radio-frequency attacks
- Navigation level attacks (modify navigation messages)
- Application hacking
- Hardware & Software tampering

The main threats to such systems are Denial of Service- (DoS), Replay- and Spoofing attacks.

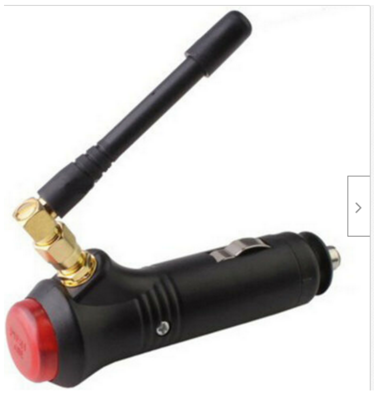
## 6.1 Denial-of-Service attacks

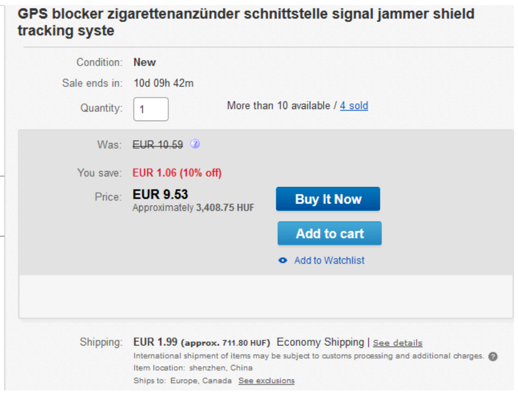
The goal of DoS attacks is to render the system unavailable for other, legitimate users. One method for GNSS DoS is to use a jammer device. A GNSS jammer device is typically a self-contained transmitter unit, sending radio signals on the same frequency as the target GNSS, interfering with the radio frequency. These can be purchased for about 10-80 euros on eBay, but it is illegal to use any type of GNSS jamming device in some countries (like the USA for example). As the price suggests, everybody could afford to purchase a device like this and it's no surprise that this is a common problem in Europe, where usage is not prohibited. The European Global Navigation Satellite System Agency launched the Strike3 program, to develop standards on GNSS threat reporting and receiver testing. As part of the program, 450000 GPS interference events were logged, of which 73000 were identified classified as a major impact on GNSS. 59000 of these major impact incidents were identified to be caused by jamming signals [10].

It's out of scope for the OSNMA system to protect against these types of DoS attacks.

## 6.2 Replay attacks

Replay attacks are a special type of man-in-the-middle attacks, specifically performed by capturing valid GNSS messages and then later retransmitting them without any modification to the actual message. By replaying a message, an adversary could avoid detection by cryptographic means while still presenting a valid GNSS signal which is not normally visible from the victim's current location. To perform such an attack, an adversary needs to be able to receive,





**GPS blocker zigarettenanzünder schnittstelle signal jammer shield tracking syste**

| | |
|---|---|
| Condition: | New |
| Sale ends in: | 10d 09h 42m |
| Quantity: | 1 — More than 10 available / 4 sold |
| Was: | ~~EUR 10.59~~ |
| You save: | EUR 1.06 (10% off) |
| Price: | EUR 9.53 Approximately 3,408.75 HUF |
| Shipping: | EUR 1.99 (approx. 711.80 HUF) Economy Shipping \| See details |

International shipment of items may be subject to customs processing and additional charges.
Item location: shenzhen, China
Ships to: Europe, Canada See exclusions

Figure 21: A GPS jammer which can be plugged into a car's cigarette lighter socket. Available to buy for under 10 euros on ebay.

store and replay the captured GNSS messages (with some added delay). Signal capturing and replaying can be done at the message or symbol level. Replaying devices on the market vary in capabilities and price, but over the years these replaying devices became cheaper. Today, USB3 to VGA adapters, which contain the FL2000 chip, are capable of replaying GNSS signals. If someone owns an Android smartphone they can use an application called GPS Replay to record and replay GPS navigation messages, fooling the other applications on the phone. Although it'd be difficult to perform a successful attack with this type of hardware, this shows how widespread the replaying functionalities have become over the years. These simple attacks are so-called meaconing attacks, but they are easily detectable with the deployment of cryptographically secured code which contains unpredictable patterns.

A naive hack would be to replace the unpredictable patterns with dummy values. This approach may work on some receivers which don't check the unpredictable patterns to detect replayed traffic, but against more elaborate receiver implementations this approach obviously would fail on the cryptographic verification tests.

Secure Code Estimate-Replay (SCER) is a more advanced type of replay attack which tries to circumvent unpredictable patterns by guessing them. However, rather than attempting to break the underlying cryptographic methods, SCER operates on the physical layer. Consider $s_n$ to be the sequence of unpredictable patterns, where the elements are assumed to be independently, but identically distributed random bits. SCER continuously tries to come up with a likely estimation for the pattern based on the observed interval. This process requires a much more involved adversary than what's seen in meaconing attacks. The cost of executing an attack like this is also much higher.

It's possible to protect against replay-attacks on two levels: receiver-level protection is about interference and spoofing detection (most receivers on the market only use receiver-level protection). At the system level, the data can be defended by constructing the new signals in an elaborate way (for example OSNMA). OSNMA-capable receivers can use the random patterns in the navigation messages to implement cryptographic replay-protection.

Cross-authentication doesn't protect replay-attacks out-of-the-box either. For satellites not providing authentication, the replay attack is not possible to detect, because the plaintext information provided could be predicted or captured. To protect against replay attacks in the cross-authentication setting, the following modifications would have to be made:

- Satellites not providing authentication information should send a few pseudorandom symbols intertwined in the navigation data
- Satellites providing authentication should authenticate both the navigation data and pseudorandom symbols of the surrounding satellites

If these methods are in place, an attacker would have more trouble trying to replay the signal provided by a plaintext satellite.

## 6.3 Spoofing attacks

GNSS spoofing is similar to jamming in that both are a type of radio interference, weak signals are getting overpowered by a stronger signal on the same frequency. It's different from jamming in the sense that spoofing is a more elaborate way of interfering with the signal, causing a receiver to believe he is in some false position. During an attack, a transmitter located near the victim would send false GNSS signals to the receiver unit. This type of attack is very easy to perform on software-defined radios, where the traditional hardware components are software-based, making attacks only a question of code.

The number of spoofing incidents has increased over the years. C4ADS is a non-profit organization dedicated to the data-driven analysis and evidence-based reporting of conflict and security issues worldwide. The organization published a report that documents around 10000 cases of GPS spoofing incidents in Russia and Syria. Most of the incidents affected ships, but airports were also another hotspot.

There are two main variants of spoofing attacks:

1. Denial-of-Service Spoofing is a form of jamming attack. The attacker is mimicking authentic GNSS signals to hijack a vulnerable receiver's tracking loops and feed it blank or false positional information. In most cases, the effects of interference are visible to the user, meaning that the victim will know that something is interfering with the GNSS signal.
2. Deception Spoofing is when the attacker is mimicking authentic GNSS signals to hijack a vulnerable receiver's tracking loops and feed it false positional or timing information to covertly misdirect the receiver and its user(s) to some desired location. Attacks of this nature will deceive both the receiver unit and their users.

OSNMA makes all types of spoofing attacks more difficult to perform. The TESLA protocol's properties assure that spoofing a location is a hard task. An attacker trying to spoof OSNMA would have to obtain the TESLA keys used by the satellites for some $K_0, ..., K_n$ key chain for it to be able to send valid MACK blocks with the spoofed navigation data. The keys are announced starting from $K_0$, but due to the properties of the one-way key-generation function, it's infeasible for an attacker to invert the function and obtain the key-chain from any of the previous keys [9]. Without the key, an attacker would have to forge a truncated MAC tag for each tag present in a MACK block sent by OSNMA for either HMAC-SHA-256 or CMAC-AES. For a single 15-bit MAC, the probability of success is $\frac{1}{2^{15}}$, a spoofer would have to produce these tags multiple times on the fly, as it's sending the spoofed location. Without breaking the underlying TESLA protocol, or obtaining the key chain by some other means, this is infeasible for an attacker to do.

The TESLA protocol is known to be secure if the $F$ function used for generating the key chain is a pseudorandom function, and if there's loose time synchronization between the receiver and the transmitter: the receiver has to know an upper bound on the receiver's clock. These two attributes are both satisfied by the OSNMA system, the receiver can achieve time synchronization when receiving the first page of a Galileo subframe. When this happens, the receiver's time is synchronized with the satellite clock, the starting time of the subframe plus 30 seconds gives an upper bound for the transmitter's time (since each subframe is transmitted over 30 seconds).

$F(K_m) = trunc(K_{len}, hash(K_m||GST_m))$ used for TESLA key-generation by OSNMA is based of SHA-2 and SHA-3 constructions, both having collision resistance. These hash functions are not proven to be PRFs theoretically, but in practice they are very resistant to distinguishing attacks. Both hashes result in an $F$ function in the TESLA key-generation, which can practically be considered a PRF and it provides collision resistance.

# 7 Conclusions

In the context of GNSS, entity authentication is proposed to be considered as data origin authentication and replay detection. Replay detection is very important due to the time-of-arrival information carried by the signals which are required for position computation.

If authentication is not in place the navigational data is easy to spoof and it gets cheaper and cheaper to buy sufficient hardware for such attacks every day. A European Space Agency's initiative integrated Navigation Message Authentication into Galileo, called OSNMA. The service will be capable of providing data origin authentication, and some robustness against certain replay attacks by adding unpredictable symbols to the signal.

Unfortunately, progress regarding the integration of OSNMA has been slow, there were multiple delays in the past two semesters and the service is still not available for public observation and testing. Availability of the system is still low, which is sadly blocking the work on the OSNMA receiver implementation. The low amount of test data only helped with the implementation and testing of some of the cryptographic and data-parsing operations present in the document.

Hardware support for OSNMA is also not highly available. There were a lot of problems with finding the right device, and it seems like even the best candidates still face serious issues when processing Galileo. Pilot-tracking smartphones (like Xiaomi MI 8) are not able to continuously process OSNMA, while devices which are capable of data-tracking (like Samsung S10) are less robust and still suffer from firmware bugs.

Hopefully, some of these issues will be resolved soon by the promising release of OSNMA specific documents by the European Space Agency. These documents' release date and the OSNMA launch are now promised to happen sometime during the summer of 2021, and work on the OSNMA-capable receiver shall continue.

## References

[1] Kamil Kazmierski; Radoslaw Zajdel; Krzysztof Sośnica, GPS Solutions volume 24 (111): Evolution of orbit and clock quality for real-time multi-GNSS solutions

[2] Adrian Perrig, Ran Canetti, J. D. Tygar, Dawn Song: The TESLA Broadcast Authentication Protocol

[3] Constellation Information about the Galileo system, retrieved 2021.03.19.

[4] UseGalileo - Find a Galileo-enabled device to use today

[5] Guidelines: OSNMA implementation in smartphones

[6] Play Store: GNSS Check

[7] I. Fernández, V. Rijmen, T. Ashur, J. Simón, C. Sarto, D. Calle, S. Cancela, P. Walker, G. Seco, D. Burkey, O. Pozzobon: Galileo Navigation Message Authentication Specification for Signal-In-Space Testing – v1.1

[8] European GNSS (Galileo) Open Service: Signal-in-Space Interface Control Document

[9] Adrian Perrig, Ran Canetti, J. D. Tygar, Dawn Song: Efficient Authentication and Signing of Multicast Streams over Lossy Channels, retrieved 2021.03.26.

[10] European Global Navigation Satellite Systems Agency: Strike3 Project.

[11] C4ADS: Above Us Only Stars, Exposing GPS Spoofing in Russia and Syria

[12] National Institute of Standards and Technology: FIPS PUB 180-4: Secure Hash Standard (SHS) 2012.

[13] National Institute of Standards and Technology: FIPS PUB 202: SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions 2015

[14] H. Krawczyk, M. Bellare and R. Canetti: HMAC: Keyed-Hashing for Message Authentication, 1997.

[15] International Organization for Standardization: ISO/IEC 9797-1:2011: Information technology - Security techniques - Message Authentication Codes (MACs) - Part 1: Mechanisms using a block cipher, 2011

[16] National Institute of Standards and Technology: FIPS PUB 186-4: Digital Signature Standard (DSS) 2013

[17] GPSTest Database about devices GNSS capabilities: GPSTest Database, last accessed on 2021.04.25.

[18] European Space Agency: Tests of Galileo Open Service Navigation Message Authentication underway, 2021.02.11.

## A The TESLA protocol

The Timed Efficient Stream Loss-Tolerant Authentication (TESLA) first appeared in literature in 2002, introducing the means of a Broadcast Authentication Protocol [2]. TESLA is operating with a symmetric-key chain for its keys, with a random (private) $K_n$ seed key in the beginning. Using this $K_n$ seed key the rest of the chain can be generated (with $n$ being the length of the chain) using an $F$ one-way function. $K_0$ will become the root key of the key chain, which can be considered as the public key for TESLA. In the case of the OSNMA layer, the $F$ function can be defined as the following:

$$K_0 = F^n(K_n)$$
$$F(K_m) = trunc(K_{len}, hash(K_m||GST_m)) = K_{m-1}$$

Where $K_{len}$ is the key length, $hash$ is a cryptographic hash function and $GST_m$ is the Galileo system time at the start of 30-second interval in which the sub-frame in which the key is applied begins transmission.

Optionally, an additional $F'$ key-generation function can be used on all keys to add an extra step of difficulty when trying to break the TESLA key-chain. Essentially, the final keys instead of $K_0, K_1, ..., K_n$ will become $F'(K_i) = K'_i, \forall i \in \{0, ..., n\}$. If $F'$ is not used during the key-generation, then if an attacker obtains a $K_i$ key, then he can obtain all the previous $K_{i-1}, ..., K_0$ keys as well. This is not a concern for the security of the system, if the keys are used in order, starting from $K_1$, up to $K_{n-1}$.

After the key-chain generation, the protocol works as the following:

1. The sender publishes the $K_0$ root key (this is the broadcaster's public key).

2. The sender creates a MAC tag from the navigation data using a $K_j$ (or optionally $F'(K_j)), j \in \{1, .., n-1\}$ key from the key-chain (starting from $K_1$ and incrementing by one, in the case of OSNMA the MACK section defines which key is used), then sends this message and tag.

3. With some delay the sender transmits the key used for the creation of the MAC tag (in the case of Galileo there may even be no delay because the root keys are authenticated using a digital signature).

4. When a receiver gets a $K_j$ key it also possesses the $K_0$ root key, using the one-way $F$ function the receiver can verify the key:

   $$K'_0 = F^j(K_j)$$
   $$K'_0 =^? K_0$$

   If $K'_0$ and $K_0$ are equal, then the $K_j$ key is authenticated.

   Note: If the $K_j$ key has already been authenticated, then later verification steps can use the $K_j$ from the key-chain when verifying new keys. Instead of repeatedly checking with the root key, this observation helps to obtain good performance.

5. Using the key verified from the previous step, the MAC-tag which used the key can also be verified.

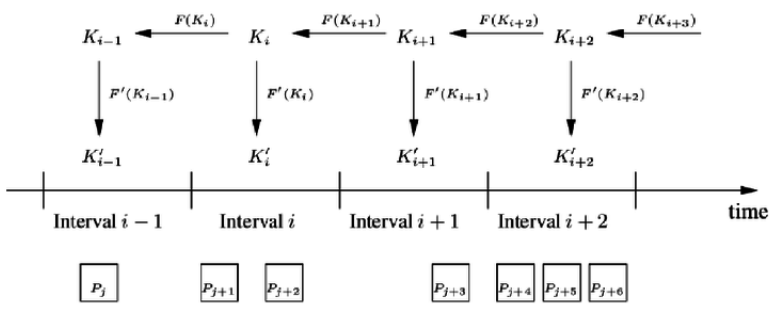


Figure 22: Illustration of the TESLA-protocol and key chain system

The TESLA protocol is secure, as long as the following assumptions are true:

- The receiver's clock is time synchronized with the sender up to a maximal error of $\delta$
- $F$ is a secure Pseudorandom Function and provides weak collision resistance

If these assumptions hold it is computationally infeasible for an attacker to forge a valid TESLA packet that the receivers will authenticate successfully. The formal security proof is presented in this paper [9].

The protocol achieves asymmetric properties by only using symmetric keys. Broadcast authentication requires an asymmetric authentication method, key distribution is not solvable on a large scale for symmetric keys. TESLA achieves this by using loose time synchronization and delayed key disclosure, making it share many attributes with asymmetric cryptographic models.

It's possible to construct a Public-Key Infrastructure (PKI) based on the TESLA protocol. If all communicating nodes are time-synchronized, then any key of the key chain is in essence a commitment to the key chain, similar to a public key of a digital signature: any loosely time-synchronized receiver with an authentic commitment can authenticate messages.