Università degli Studi di Padova

Dipartimento di Ingegneria dell'Informazione

Master Thesis in Ingegneria Informatica

A receiver centric analysis of the Galileo Open Service Navigation Message Authentication protocol

16/04/2018

Supervisor
Prof. Nicola Laurenti
Università di Padova

Master Candidate
Stefano Zanella

A.A. 2017/18

To Sarah - you're home.
To Giorgio and Lorena - you started it.

Abstract

Since its first incarnation in the 70s under the "NAVSTAR" name, satellite positioning acquired more and more relevance in the life of everyone. Directly or indirectly, we all rely on technological aids to precisely define our position. This exponential trend uncovered in recent years the need to secure the information transmitted by legitimate satellites: military signals are encrypted before transmission with schemes that allow only an authorized and restricted set of users, but for the civil case such a restriction clashes with the idea of having an open signal accessible by anyone. Yet, leaving the transmitted data unprotected opens the door to various kinds of attack.

It is with the introduction of a satellite constellation operated by the European Union under the name of Galileo that the topic of data authentication for open Global Navigation Satellite Systems (GNSS) signal took a practical turn. Some authentication schemes have been developed on paper to suit the low data rate available, resulting in the European Commission releasing an update to Galileo's Interface Control Document (ICD) to include an implementation of Timed Efficient Stream Loss-tolerant Authentication (TESLA), a protocol suited to authenticate streams of data that makes no assumption on the time when a receiver starts receiving the radio signal.

In this paper a critical analysis of the proposed specification is performed, to understand the implications on the receiver side in terms of performance and resource consumption.

Contents

Listing of figures

Listing of tables

1 Introduction

1.1 Overview

Since its introduction in 1973, GPS has seen tremendous growth in adoption. Thanks to the proliferation of use cases, positioning technologies have become the basis upon which many services are built. One of the most obvious ones is of course route guidance, the most important one arguably being aircrew assistance: GPS has played a central role in the steady improvement of critical aircraft operation reliability, such as landing and take off. A fast-growing field of application is also agriculture, which benefits from the high precision of satellite positioning by automating mechanical operations on large fields like harvesting and sowing - in many cases tractors are today totally unmanned, relying solely on the precision of GPS for covering the field of operation with high efficiency.

But the real explosion of GPS arguably started when major manufacturer started to embed receivers in smartphone and tablets. As of today, it's given for granted that any sufficiently evolved digital device is capable of decoding a GPS signal and estimate its position. This opened the door to a complete new set of applications of which the full span has yet to be seen. From routing applications to local recommendations engine, none of them would have much impact without the possibility of locating the user offered by GPS. A new field of application that's capturing the interest of major companies is the automotive one:

whether self-driving car will be soon a common reality, or whether safety concerns and regulations will push this reality far in the future, it's undeniable that GPS plays a central role in the definition of this new reality.

Following the success of the American experiment, other global powers decided to build their own version of GPS. As of today Russia, China, European Union, India and Japan operate their own satellite constellation for positioning purposes - even if each to a different degree of platform maturity, stability, scale and reach.

Russian GLONASS is the oldest of these newer constellations, and reaching a precision close to that of GPS. The Chinese BeiDou, Indian IRNSS and Japanese QZSS are all considered as of now regional systems, meaning they only cover a limited area around their respective country of operation.

Possibly the most promising development is provided by the European Galileo constellation, the youngest of the systems mentioned so far. With a nominal accuracy of 1 meter for civil signal it promises to outperform the current GPS capabilities. This constellation is currently operational with 14 satellites, with a planned total of 30; at the same time, research is still being held under many aspects of the system.

No matter the size and age of these systems, overall satellite positioning technology has seen various steps of improvements over the years, and in two main areas: hardware improvements on the space vehicles and optimizations in the usage of the allocated radio frequencies. GPS space vehicles have been continuously modernized, resulting in the release of 7 different satellite versions (blocks I to IIIA), and another 2 planned but not yet scheduled (IIIB and IIIC). Improvements generally range across improvements of nominal life, increases in the radio transmission performance (e.g. signal power at Earth's surface) and hardware additions to better cope with errors (e.g. addition of laser retro-reflectors in block IIIA to disentangle clock errors from ephemeris errors).

Another reason for the development of new space vehicles is to allow deployment of new signals. For example, the last planned block of GPS satellites is going to introduce 3 new signals (L1C, L2C and M). Each of them improves on the existing ones allowing for increased robustness against interference and data loss, and higher accuracy through increased bandwidth, longer spreading codes and

higher power.

Together with this, recent developments have also happened in the direction of enhancing satellite positioning by using other, more local sources. For example, aviation has stringent needs in terms of monitoring and warning clients in case of, e.g., clock problems. In such cases aircrafts must be notified in real-time that they can't rely on a GNSS for precision approach. Neither GPS nor GLONASS support this use case, so a support system was developed under the name of GBAS (Ground-Based Augmentation System). Aside from this, a cluster of other augmentation systems with similar purposes have been deployed; such systems go under the name of SBAS (Satellite-Based Augmentation System).
More recently, smartphones have begun to support augmentation of the satellite signal through both WiFi and cell tower triangulation, resulting in better user experiences in terms of faster and more precise resolution of the user's position.

Despite all these efforts to improve on the original GPS signal, little to no practical development has so far been made in making the civil signal secure. Current signals position themselves on two very far ends of the security spectrum. On one side, military signals provide security through encryption; this not only provides state-of-the-art security to the signal, but also allows for restricted access to it: only receivers with authorized access are able to decode the signal. On the other side of the spectrum, use of civil signals is by nature open to everyone, but as of today there is no mechanism to guarantee the authenticity of the signal itself. Malicious users can choose from a wide range of spoofing and replay attacks, allowing for almost undisturbed disruption of service. While letting a user think they're located in a remote island while being in the middle of a most crowded city might appear like a practical joke, even a subtle change in the position or velocity of an aircraft might have catastrophic consequences. Even without putting lives at risk, attacks that aim at drifting a receiver's clock can be a component of more complex network attacks - as many security algorithms rely on coarsely precise clocks for avoiding replay attacks. If nothing else, a whole class of denial of service attacks can be conceived exploiting the vulnerability of today's civil positioning technologies.

In recent years this topic gained a strong momentum, and research on different

levels and different directions has been performed to solve the problem of securing GNSS signals. Proposed solutions can be generally categorized based on what part of the GNSS signal they aim to protect. A number of proposals have been developed to make the spreading codes more robust, for example to prevent from selective-delay attacks. These techniques seem to provide better overall security but are arguably more demanding on the receiver side in terms of implementation. A simpler general approach is instead that of leaving the overall signal structure unchanged, protecting only the navigation message carried by the signal. It is this last approach that has been currently taken to protect Galileo's public regulated service in the form of the TESLA protocol.

The TESLA protocol is an asymmetric data authentication protocol specifically designed to work over a continuous, real-time stream of data over a lossy channel. The main idea behind it is that of using hash chains to generate message authentication codes, with keys being revealed periodically to allow for verification of previously-received messages. In a recent specification, the theoretical protocol has been adapted to fit the current Galileo data signal by taking advantage of the reserved bit blocks factored into the actual frame structure, which allows for support over the E1b and E1c signals.

As of today, the protocol remains in form of a separate specification from the ICD, but in the course of 2018 it's supposed to be part of the Signal in Space (SIS) testing. The goal of this publication is to provide a software implementation of this scheme that follows the guidelines approved by ESA, to allow for easy but accurate testing of the operational parameters that are going to be subject to change and/or remain variable in the final implementation. This effort joins forces with the more-GOSSIP project being developed at the Department of Electrical Engineering of the Università degli Studi di Padova. This project aims at developing a software simulator that can test a number of possible attacks on Galileo (and GPS) signals, and implementation of the proposed Galileo OSNMA (Open Service Navigation Message Authentication) is part of it.

1.2 Outline

This document starts by giving a detailed panorama of how radio navigation works; in Chapter 2 we will describe how a receiver can determine its position, velocity and time by using information received from 4 or more satellites. We'll proceed in Chapter 3 to give a brief account of the evolution of navigation systems that will put Galileo in the right historical context. Next, in Chapter 4 we'll explore TESLA, an authentication protocol suited for data streams and OSNMA, the Galileo implementation of TESLA. Chapter 5 is dedicated to analyze the steps a receiver needs to perform to reach the state in which received data can be authenticated continuously. In Chapter 6 we'll continue the analysis of OSNMA and provide clarity over certain aspects that might concern receiver implementors. Finally, in Chapter 7 a summary of the findings and possible future expansions is given.

2

Satellite navigation fundamentals

No matter the GNSS under analysis, the fundamental principles that make radio satellite navigation work are the same. In this chapter we'll describe the equations that make satellite navigation a reality; understanding these mechanics will be of importance when talking about the challenges of implementing TESLA on the receiver side.

2.1 Overview of ranging through TOA measurements

To introduce the problem of satellite navigation, let's assume a generic actor at position P wanting to know its position in a three-dimensional coordinate frame. Let's assume also the presence of a beacon, whose position T is fixed and precisely known. This beacon transmits messages at regular intervals, which the actor is capable of intercepting. These messages contain two pieces of information: the precise time t_0 at which the message was sent, and a description of T, i.e. the position of the beacon at the time of transmission. Let's also assume that the transmission happens over a medium with a specific speed of propagation s determined by the law of physics. Finally, let's assume both actor and beacon are equipped with perfectly synchronized clocks, so that locally observed time for each party is equal to that of the other counterpart.

Given these assumptions the actor has all the information needed to determine

its position with respect to the beacon. The resolution of this problem is based on the Time Of Arrival (TOA) of the message. If the actor marks the time t_1 at which the message arrives, then its distance from the beacon can be computed by multiplying the flight time of the message $t_1 - t_0$ by its travelling speed s. The resulting range measurement r_1 defines a sphere of radius r_1 centered at the position of the beacon T, and the actor position is a point of this sphere.

To further narrow down the precise location of the actor P, it's sufficient to have another two beacons at positions different than T. With the same technique, the actor can determine its distance from the other two beacons. Determining its distance r_2 from the second beacon will mean its position is on a sphere of radius r_2 centered around the beacon. Intersecting this sphere with the sphere around the first beacon will yield a circumference (or, in its degenerate case, a point if the two spheres are tangent - a condition that will be ignored for now as it's not the general case). Repeating the measurement with the third beacon will also again yield a sphere of radius r_3 that can be intersected with the other two, resulting in the identification of two points, one being the actor's true position and the other being its mirror with respect to the beacons' plane. In order to uniquely determine its position, the actor can either perform a fourth measurement, or introduce other constraints in the equation that automatically exclude one of the two points. For example, for users on the Earth's surface the point with lower altitude can be selected with no further measurement.

This contrived example forms the basic of triangulation through TOA measurement, which is at the base of GNSS. In the real world, though, the hypotesis that allowed the problem to be solved in such simple terms is never true, and using this simplified set of constraints is not enough to provide precise positioning. Therefore, more relaxed constraints and more complex models are used as a basis for satellite navigation, which we'll describe next. Nevertheless, the fundamental concept remains unchanged even with a more elaborated model of the world.

2.2 Determining satellite-to-user range

Improving on the model just described, the first assumption we'll refine is that of the clock of the receiver being perfectly synchronzied with that of the transmitter. In the model for PVT calculation there are many factors that can influence the

final result, but hardly any of those can have an impact as big as that of an imprecise time calculation.

Before talking about how unsynchronized clocks affect the position resolution, let's define the general model used by GNSS.

As saw already, the model for PVT calculation includes two clocks: that of the transmitter and that of the receiver. With differences in precision and amount of drift, both clocks are assumed to accumulate (positive or negative) drift with respect to system time, thus being offset from the reference time. We'll call the offset of the transmitter's clock δ_t and the offset at the receiver's δ_r.

Recalling that the geometric distance r between transmitter and receiver is

$$r = c(T_r - T_t) \tag{2.1}$$

where

- c is the speed of light at which the signal travels
- T_r is the system time of signal reception
- T_t is the system time of signal transmission

we can then improve the equation by including the time offsets just described, which gives us the equation for the *pseudorange* ρ:

$$\begin{aligned} \rho &= c[(T_r + \delta_r) - (T_t + \delta_t)] \\ &= c[(T_r + \delta_r) - (T_t + \delta_t)] \\ &= c(T_r - T_t) + c(\delta_r - \delta_t) \\ &= r + c(\delta_r - \delta_t) \end{aligned} \tag{2.2}$$

Of these two offset, that of the satellite is monitored and controlled by each GNSS ground mission, and its correction transmitted as part of the navigation message. Therefore, despite some residual offset its contribution to the equation can be ignored, thus simplifying the model for the pseudorange to

$$\rho = r + c(\delta_r) \tag{2.3}$$

This transforms the original problem from having three unknowns to having four.

2.3 Determining user's position

Solution of equation 2.3 yields the magnitude of a vector that represents the distance between transmitter and receiver. In reality, what we're interested in is identifying the coordinates of the receiver within a reference frame. To model this we can transform r in cartesian form. Given a receiver at position (x_r, y_r, z_r) and a transmitter at position (x_t, y_t, z_t), the pseudorange can be expressed as:

$$\begin{aligned} \rho &= \sqrt{(x_t - x_r)^2 + (y_t - y_r)^2 + (z_t - z_r)^2} + c\delta_r \\ &= f(x_r, y_r, z_r, \delta_r) \end{aligned} \tag{2.4}$$

Given that the problem contains four variables, we need a set of four equations to obtained a closed solution. This is equivalent of obtaining ranges from four different satellites, which can be expressed with the set of equations

$$\begin{aligned} \rho_1 &= \sqrt{(x_{t1} - x_r)^2 + (y_{t1} - y_r)^2 + (z_{t1} - z_r)^2} + c\delta_r \\ \rho_2 &= \sqrt{(x_{t2} - x_r)^2 + (y_{t2} - y_r)^2 + (z_{t2} - z_r)^2} + c\delta_r \\ \rho_3 &= \sqrt{(x_{t3} - x_r)^2 + (y_{t3} - y_r)^2 + (z_{t3} - z_r)^2} + c\delta_r \\ \rho_4 &= \sqrt{(x_{t4} - x_r)^2 + (y_{t4} - y_r)^2 + (z_{t4} - z_r)^2} + c\delta_r \end{aligned} \tag{2.5}$$

This set of nonlinear equation can be solved with different techniques: closed-form solutions, iterative methods based on linearization, or Kalman filters. What follows is an example of linearization.

To transform non-linear terms in the equation, we will make the assumption that an approximate position $(\hat{x}_r, \hat{y}_r, \hat{z}_r)$ and clock error estimate $\hat{\delta}_r$ are available. Under this assumption we can then say that the true receiver position and time are composed of an approximate component and an incremental one:

$$\begin{aligned} x_r &= \hat{x}_r + \Delta x_r \\ y_r &= \hat{y}_r + \Delta y_r \\ z_r &= \hat{z}_r + \Delta z_r \\ \delta_r &= \hat{\delta}_r + \Delta \delta_r \end{aligned} \tag{2.6}$$

This allows us to express 2.4 as

$$f(x_r, y_r, z_r, \delta_r) = f(\hat{x}_r + \Delta x_r, \hat{y}_r + \Delta y_r, \hat{z}_r + \Delta z_r, \hat{\delta}_r + \Delta \delta_r) \tag{2.7}$$

This can in turn be expaneded using a Taylor series as

$$\begin{aligned} f(\hat{x}_r + \Delta x_r, \hat{y}_r + \Delta y_r, \hat{z}_r + \Delta z_r, \hat{\delta}_r + \Delta \delta_r) = f(\hat{x}_r, \hat{y}_r, \hat{z}_r, \hat{\delta}_r) + \\ \frac{\partial f(\hat{x}_r, \hat{y}_r, \hat{z}_r, \hat{\delta}_r)}{\partial \hat{x}_r} \Delta x_r + \\ \frac{\partial f(\hat{x}_r, \hat{y}_r, \hat{z}_r, \hat{\delta}_r)}{\partial \hat{y}_r} \Delta y_r + \\ \frac{\partial f(\hat{x}_r, \hat{y}_r, \hat{z}_r, \hat{\delta}_r)}{\partial \hat{z}_r} \Delta z_r + \\ \frac{\partial f(\hat{x}_r, \hat{y}_r, \hat{z}_r, \hat{\delta}_r)}{\partial \hat{\delta}_r} \Delta \delta_r + \\ \ldots \end{aligned} \tag{2.8}$$

The expression has been truncated to eliminate higher order non-linear components. Focusing on the partial derivatives we obtain

$$\begin{aligned} \frac{\partial f(\hat{x}_r, \hat{y}_r, \hat{z}_r, \hat{\delta}_r)}{\partial \hat{x}_r} &= -\frac{x_t - \hat{x}_r}{\hat{r}_t} \\ \frac{\partial f(\hat{x}_r, \hat{y}_r, \hat{z}_r, \hat{\delta}_r)}{\partial \hat{y}_r} &= -\frac{y_t - \hat{y}_r}{\hat{r}_t} \\ \frac{\partial f(\hat{x}_r, \hat{y}_r, \hat{z}_r, \hat{\delta}_r)}{\partial \hat{z}_r} &= -\frac{z_t - \hat{z}_r}{\hat{r}_t} \\ \frac{\partial f(\hat{x}_r, \hat{y}_r, \hat{z}_r, \hat{\delta}_r)}{\partial \hat{\delta}_r} &= c \end{aligned} \tag{2.9}$$

where

$$\hat{r}_t = \sqrt{(x_t - \hat{x}_r)^2 + (y_t - \hat{y}_r)^2 + (z_t - \hat{z}_r)^2} \tag{2.10}$$

At this point we can substitute 2.4 and 2.9 into 2.8, which yields (for a generic range between transmitter j and receiver r):

$$\rho_j = \hat{\rho}_j - \frac{x_j - \hat{x}_r}{\hat{r}_j} \Delta x_r - \frac{y_j - \hat{y}_r}{\hat{r}_j} \Delta y_r - \frac{z_j - \hat{z}_r}{\hat{r}_j} \Delta z_r + c\delta_r \tag{2.11}$$

This completes the linearization of 2.4 with respect to the unknown $\Delta x_r, \Delta y_r, \Delta z_r, \Delta \delta_r$. For clarity we can rearrange unknowns on the right and knowns on the left

$$\hat{\rho}_j - \rho_j = \frac{x_j - \hat{x}_r}{\hat{r}_j} \Delta x_r + \frac{y_j - \hat{y}_r}{\hat{r}_j} \Delta y_r + \frac{z_j - \hat{z}_r}{\hat{r}_j} \Delta z_r - c\delta_r \tag{2.12}$$

and then introduce new variables

$$\begin{aligned} \Delta\rho &= \hat{\rho}_j - \rho_j \\ a_{xj} &= \frac{x_j - \hat{x}_r}{\hat{r}_j} \\ a_{yj} &= \frac{y_j - \hat{x}_r}{\hat{r}_j} \\ a_{zj} &= \frac{z_j - \hat{z}_r}{\hat{r}_j} \end{aligned} \tag{2.13}$$

which allows us to rewrite the equation as

$$\Delta\rho_j = a_{xj}\Delta x_r + a_{yj}\Delta y_r + a_{zj}\Delta z_r - c\delta_r \tag{2.14}$$

Now the original formulation of the problem stated in 2.5 can be rewritten in linear form as

$$\begin{aligned} \Delta\rho_1 &= a_{x1}\Delta x_r + a_{y1}\Delta y_r + a_{z1}\Delta z_r - c\delta_r \\ \Delta\rho_2 &= a_{x2}\Delta x_r + a_{y2}\Delta y_r + a_{z2}\Delta z_r - c\delta_r \\ \Delta\rho_3 &= a_{x3}\Delta x_r + a_{y3}\Delta y_r + a_{z3}\Delta z_r - c\delta_r \\ \Delta\rho_4 &= a_{x4}\Delta x_r + a_{y4}\Delta y_r + a_{z4}\Delta z_r - c\delta_r \end{aligned} \tag{2.15}$$

These equations can then be put in matrix form; by defining

$$\Delta\boldsymbol{\rho} = \begin{bmatrix} \Delta\rho_1 \\ \Delta\rho_2 \\ \Delta\rho_3 \\ \Delta\rho_4 \end{bmatrix} \quad \boldsymbol{H} = \begin{bmatrix} a_{x1} & a_{y1} & a_{z1} & 1 \\ a_{x2} & a_{y2} & a_{z2} & 1 \\ a_{x3} & a_{y3} & a_{z3} & 1 \\ a_{x4} & a_{y4} & a_{z4} & 1 \end{bmatrix} \quad \Delta\boldsymbol{x} = \begin{bmatrix} \Delta x_r \\ \Delta y_r \\ \Delta z_r \\ -c\Delta\delta_r \end{bmatrix}$$

we can then write

$$\Delta\boldsymbol{\rho} = \boldsymbol{H}\Delta\boldsymbol{x} \tag{2.16}$$

which has solution

$$\Delta\boldsymbol{x} = \boldsymbol{H}^{-1}\Delta\boldsymbol{\rho} \tag{2.17}$$

Once the unknown are computed, the actual user position can be computed using 2.6. This process works well as long as the displacement from the approximate position is small enough. However, subsequent iterations of the method will improve the accuracy of the result.

In reality though, the true user-to-satellite range measurements are affected by independent errors such as measurement noise and multipath. Such errors con-

stitute a component that affects the calculated user position. To minimize these errors, one technique is to perform measurements to more than four satellites, using least square estimation methods to minimize the effect of the independent source of errors. This is a feature that most of today's user receivers employ.

2.4 Determining user's velocity

Another component of the PVT solution is velocity. Satellite navigation provides the capability of measuring a receiver's three-dimensional velocity, here denoted as $\dot{\boldsymbol{u}}$. This is generally determined using one of two techniques: either as an approximate derivative of the receiver position, taking the difference of two subsequent measurements and dividing it by the time elapsed between the two. This technique can provide satisfactory results only if the receiver's velocity is nearly constant and if the errors in determining the two positions are small enough. For this reason most modern receivers use a technique based on measurement of the carrier frequency and a precise estimation of the received Doppler frequency.

Generally speaking, the received frequency at the receiver's antenna can be approximated by the classical Doppler equation:

$$f_R = f_T \left(1 - \frac{\boldsymbol{v_r} \cdot \boldsymbol{a}}{c}\right) \tag{2.18}$$

where f_T is the transmitted frequency at the satellite's end, f_R is the received frequency at the receiver's antenna, c is the speed of light, $\boldsymbol{a}$ is the unit vector directed along the line of sight between satellite and receiver, and $\boldsymbol{v_r}$ is the satellite-to-receiver relative velocity vector. Given $\boldsymbol{v}$ as the satellite's velocity, we can express $\boldsymbol{v_r}$ as

$$\boldsymbol{v_r} = \boldsymbol{v} - \dot{\boldsymbol{u}} \tag{2.19}$$

This relative motion results in a Doppler offset that can be expressed by

$$\Delta f = f_R - f_T = -f_T \frac{(\boldsymbol{v} - \dot{\boldsymbol{u}}) \cdot \boldsymbol{a}}{c} \tag{2.20}$$

At this point several techniques can be used to determine $\dot{\boldsymbol{u}}$ from the received Doppler frequency. The one described here assumes that the user position has already been determined and computes, together with the receiver's velocity $\dot{\boldsymbol{u}} =$

$(\dot{x}_u, \dot{y}_u, \dot{z}_u)$, also the clock drift $\dot{t}_u$.

We start by substituting 2.19 into 2.18, which yields

$$f_{Rj} = f_{Tj}\left\{1 - \frac{1}{c}\left[(\boldsymbol{v}_j - \dot{\boldsymbol{u}}) \cdot \boldsymbol{a}_j\right]\right\} \tag{2.21}$$

At this point the receiver needs to estimate the received signal frequency. This measurement is also subject to error due to the bias induced by drifting of the local clock, expressed by $\dot{t}_u$. Hence, being f_j the estimate of the received frequency, we can put clock drift, actual and estimated received frequency in the following relation

$$f_{Rj} = f_j(1 + \dot{t}_u) \tag{2.22}$$

Substituting 2.22 into 2.21 and performing some basic algebraic manipulation we obtain

$$\frac{c(f_j - f_{Tj})}{f_{Tj}} + \boldsymbol{v}_j \cdot \boldsymbol{a}_j = \dot{\boldsymbol{u}} \cdot \boldsymbol{a}_j - \frac{cf_j\dot{t}_u}{f_{Tj}} \tag{2.23}$$

if then we expand the dot products using the three-dimensional components of the vectors we reach

$$\frac{c(f_j - f_{Tj})}{f_{Tj}} + v_{xj}a_{xj} + v_{yj}a_{yj} + v_{zj}a_{zj} = \dot{x_u}a_{xj} + \dot{y_u}a_{yj} + \dot{z_u}a_{zj} - \frac{cf_j\dot{t}_u}{f_{Tj}} \tag{2.24}$$

In this expression all variables on the left side are either calculated or derived from measured values. To make the equation more compact we can then introduce a new variable

$$d_j = \frac{c(f_j - f_{Tj})}{f_{Tj}} + v_{xj}a_{xj} + v_{yj}a_{yj} + v_{zj}a_{zj} \tag{2.25}$$

Additionally, the term f_j/f_{tj} is tipically very close to 1 and the resulting error is negligible. With this further simplification we obtain

$$d_j = \dot{x_u}a_{xj} + \dot{y_u}a_{yj} + \dot{z_u}a_{zj} - c\dot{t}_u \tag{2.26}$$

Now the equation contains four unknowns which can be solved by performing four distinct measurements against four different satellites, yielding a system of

equations that can represented and solved with the following matrices:

$$\boldsymbol{d} = \begin{bmatrix} d_1 \\ d_2 \\ d_3 \\ d_4 \end{bmatrix} \quad \boldsymbol{H} = \begin{bmatrix} a_{x1} & a_{y1} & a_{z1} & 1 \\ a_{x2} & a_{y2} & a_{z2} & 1 \\ a_{x3} & a_{y3} & a_{z3} & 1 \\ a_{x4} & a_{y4} & a_{z4} & 1 \end{bmatrix} \quad \boldsymbol{g} = \begin{bmatrix} \dot{x}_u \\ \dot{y}_u \\ \dot{z}_u \\ -c\dot{t}_u \end{bmatrix}$$

which then corresponds to

$$\boldsymbol{d} = \boldsymbol{H}\boldsymbol{g} \tag{2.27}$$

so the solution for the user's velocity and time drift is

$$\boldsymbol{g} = \boldsymbol{H}^{-1}\boldsymbol{d} \tag{2.28}$$

As in the previous case, this solution is affected by errors propagating across the calculation. Also similar to the previous case, it's possible to perform measurements against more than four satellites and use least square estimates to increase the accuracy.

2.5 Terrestrial reference frame

So far in the discussion of how to resolve user's PVT equation we took for granted that a coordinate reference frame was provided. We'll now look into how this coordinate reference frame is modeled and how receivers can map between this reference frame and a pair of longitude and latitude estimates.

In general, two kind of models can be used: inertial and rotating. When describing the orbit of satellites, it's more useful to use an intertial system; in particular, an Earth Centered Inertial (ECI) system has origin at the center of mass of the Earth and axes that point in fixed directions with respect to the stars.

When dealing with measuring and describing user position, instead, it's more useful to use a reference system that's fixed with respect to the Earth (i.e. a reference system that rotates with it). Such a system is called Earth Centered Earth Fixed (ECEF). In such a system the xy-plane is coincident with Earth's equatorial plane: the x axis pointing to 0° longitude and the y axis pointing to 90°E longitude. The z axis is chosen to be normal to the equatorial plane in the direction of the geographical North Pole, thus forming a right-handed coordinate

system.

These frames are useful for describing and solving the PVT problem, but are less so when a fix needs to be presented to the user. This normally is better described by a pair of longitude and latitude measurements. Therefore in order to make this transformation, a physical model of the Earth is needed. Such model is not unique and varies between satellite systems; Galileo uses what's known as GTRF (Galileo Terrestrial Reference Frame).

The GTRF is an independent realization of an international standard called ITRS (International Terrestrial Reference System) which defines a framework to describe geodetic reference frames. This framework is used to produce periodic updates to the ITRF (International Terrestrial Reference Frame); by Galileo system requirement the three-dimensional difference between a position in GTRF and the most recent ITRF should not exceed 3cm.

The purpose of such a reference frame is to define the Earth's surface by a closed equation representing an ellipsoid with a certain number of degrees of freedom. These degrees of freedom are then represented by parameters that can be sent along the navigation message and subsequently used to resolve user's position in terms of longitude, latitude and height with respect to this reference frame; defined in this manner, these parameters are called *geodetic*.

Given a user's position vector $\boldsymbol{u} = (x_u, y_u, z_u)$ in ECEF coordinates, we can compute the user's geodetic longitude λ as the angle between the user and the x axis measured in the xy-plane:

$$\lambda = \begin{cases} \arctan\left(\frac{y_u}{x_u}\right), & \text{if } x_u \geq 0 \\ 180 + \arctan\left(\frac{y_u}{x_u}\right), & \text{if } x_u < 0 \text{ and } y_u \geq 0 \\ -180 + \arctan\left(\frac{y_u}{x_u}\right), & \text{if } x_u < 0 \text{ and } y_u < 0 \end{cases} \tag{2.29}$$

Geodetic latitude ϕ and height h are defined in terms of the normal to the ellipsoid from the user's position $\boldsymbol{n}$. Geodetic height is defined as the minimum distance of the user to the surface of the reference ellipsoid. Geodetic latitude is the angle between the normal $\boldsymbol{n}$ and its projection into the equatorial plane. Conventionally ϕ is set to be > 0 if the user is in the northern empisphere ($z_u > 0$) and < 0 if the user is in the southern emisphere ($z_u < 0$).

To calculate ϕ and h several methods exist, both closed-form and iterative; one

example of the latter is a highly convergent method by Bowring which uses four parameters in the description of the ellipsoid:

- a semimajor axis, i.e. radius of the cross-section taken at the equatorial plane
- b semiminor axis, i.e. radius of the polar section of the Earth
- e eccentricity, defined as
$$e = \sqrt{1 - \frac{b^2}{a^2}}$$
- e' second eccentricity, defined as
$$e' = \sqrt{\frac{a^2}{b^2} - 1} = \frac{a}{b}e$$

The resolution method, given these parameters, is as follows:

$$\begin{aligned}
&p = \sqrt{x^2 + y^2} \\
&\tan u = \left(\tfrac{z}{p}\right)\left(\tfrac{a}{b}\right) \\
&\text{iteration loop} \\
&\cos^2 u = \tfrac{1}{1+\tan^2 u} \\
&\sin^2 u = 1 - \cos^2 u \\
&\tan\phi = \tfrac{z+e'^2 b \sin^3 u}{p - e^2 a \cos^3 u} \\
&\tan u = \tfrac{b}{a}\tan\phi \\
&\text{until } \tan u \text{ converges, then} \\
&N = \tfrac{a}{\sqrt{1-e^2\sin^2\phi}} \\
&\text{if } \phi \neq \pm 90^\circ \\
&h = \tfrac{p}{\cos\phi} - N \\
&\text{otherwise if } \phi \neq 0 \\
&h = \tfrac{z}{\sin\phi} - N + e^2 N
\end{aligned}$$

2.6 Additional sources of errors

So far we considered as a potential source of error just the clock drift at the receiver's end, provided the drift at the satellites' end is corrected by the ground segment and the correction parameters transmitted with the navigation message.

The reason for this is that this source of error has an impact that makes any other error less than significant; nevertheless, once clock drift has been accounted for, other sources of error can become relevant and affect the accuracy of the final fix.

These error sources include:

- Ephemeris errors: these happen as the ephemerides transmitted by ground mission are a prediction of the actual trajectory. In the case of Galileo, differences between the actual orbit path and the broadcasted one can introduce errors in pseudorange measurements in the order of 1.6m $\pm$ 0.3m.
- Atmospheric errors: these are of different kind. The more general one is due to the dispersion effect that happens in mediums with an index of refraction that is a function of the wave frequency (as is in the case of Earth's atmosphere). This results in different propagation speeds between signal phase and information. Another kind of error is due to the crossing of the signal of Earth's ionosphere; the effect is similar to the general atmospheric propagation delay, but its magnitude is strong enough that correction parameters are provided as part of the navigation message. Since this error is frequency-dependent, another way to compensate for this effect is to use a dual-frequency receiver, making measurements on two separate frequency bands.
- Relativistic effects: the satellites' clocks are affected by both general relativity and special relativity laws. Galileo OS SIS ICD doesn't make a specific statement with reference to relativistic effects, but provides nevertheless four clock correction parameters to compensate for various aspects that might influence the frequency of the onboard clocks.

3

A brief history of GNSS

In order to understand the context in which the problem of data authentication for Galileo is studied, it's useful to know how the major GNSS constellations evolved and what are the traits that characterize them at present time. In this chapter we'll briefly introduce Galileo, GPS, and the other main players in the satellite positioning arena.

3.1 Global Positioning System (GPS)

In February 1978, the first satellite of the Block I family was put in orbit by the U.S. government. It was the first of a series of 11 satellites that will constitute the prototype of what's currently known as GPS. This was the first concrete realization of what would soon be world's most used radio navigation system, and was the result of putting together ideas and prototypes that had been developed by several U.S. military branches.

3.1.1 Overview

As of today, GPS is comprised of three segments: the space segment, ground segment and user segment.

The space segment is the constellation of satellites that broadcast the radio signal employed by users to determine their positions. The ground segment is the

operating network in charge of maintaining satellites in their proper orbit and of making sure the data that's sent is accurate. The user segment is composed of user receiving equipment. GPS receivers are mainly responsible for resolving the PVT equation after having acquired and tracked a sufficient number of satellites. In addition to that, GPS receivers can also offer other kinds of measurements - like platform attitude (heading, pitching and roll) - or function as precise sources of time.

3.1.2 Ground segment

Fundamental to the correct operation of GPS is the health of the satellite constellation and the accuracy of the navigation message. These two tasks are the main responsibility of the ground segment - which includes a redundant Master Control Station (MCS), monitoring stations (MS) and ground antennas (GA). The MCS performs various functions like generating the navigation message, monitoring SVs health and performing housekeeping operations on them, and maintaining time synchronization against UTC.

In order to do so, the MCS communicates through satellites via GAs, which are deployed across the globe so that each satellite can be reached by one to four antennas depending on its position. A GA is a full-duplex S-band communication facility that can hold a dedicated control and command session with a SV, operating under control of the MCS.

To receive precise information over the constellation's health, the MCS also operates a set of MSs that is capable of receiving GPS signal over the full L-band and to send this signal together with meteorological data back to the MCS. In addition to that, MSs include as part of their equipment high precision redunant cesium clocks, which are used as a reference to measure precision of the received data.

3.1.3 Space segment

The original design of GPS includes 24 satellites, divided in six orbital planes positioned approximately 26 600km from the mass center of Earth. The nominal orbital period is one-half of a sidereal day, or 11 hours and 58 minutes. This design

allows for global coverage, meaning that at nominal constellation operativity a minimum of three satellites is visible at any position on Earth's surface.

Space vehicles (SVs) had undergone several modernizations steps starting from the first launch in 1973. Subsequent developments are organized into blocks, Block I being the first. The current constellation includes SVs from blocks IIR, IIR-M and IIF, with a new block IIIA being planned for the next years.

Nevertheless, the main components of a SV remain the same: an antenna farm, antennas for communication with the ground segments, a solar array for providing energy to the satellite, a propulsion engine for orbital adjustments, high-precision rubidium clocks, memory storage for robust operation under loss of link with the ground segment, and of course a computing unit. Each of these main building blocks are subject of the modernization efforts, with the goal of making each new block more robust to adverse oribting conditions, more tolerant to failures, more flexible to changing requirements, and more precise. At the same time, every improvement contributes to achieve higher SV longevity: as an example the design life of a Block I satellite was 5 years, while that of a Block IIF satellite is 12 years.

3.1.4 Signal features

Modernization of satellites doesn't happen only because of hardware improvements, but also because the GPS specification is in constant evolution. The current GPS signal plan comprises transmission on three separate bands: L1, L2 and L5. The former two bands are part of the original plan, which included a C/A (coarse acquisition) signal on L1 and a military signal named P(Y) on both L1 and L2. Modernization of the signal plan introduced three new civil signals, one on L1 (L1C), one on L2 (L2C) and one on a new band L5 which will also support SoL (Safety of Life) operations. In addition to this, a new military signal (M) has been added on the L1 and L2 bands.

Originally, the GPS signal used a BPSK modulation that in conjunction with the use of spreading codes allowed for maximization of bandwidth usage. To accomodate for transmission of different signals on the same frequency band, BOC modulation has been introduced in newer signals. This achieves sufficient spectral separation for existing receivers to keep working and for new receivers to

be able to isolate each transmission. This idea has been also a major driver in the Galileo's signal plan as explained later in this chapter.

3.2 Galileo

Following the initiative of the U.S. government and that of the former Russian federation, in 1998 the European Union decided to develop its own satellite constellation to offer positioning services. Despite the challenge of being developed as part of a program that bonds together 27 member states with individual agendas and priorities, Galileo is set to mark a milestone in the evolution of positioning services. Being the youngest of the global constellations, it builds on top of the lessons learned during four decades of GPS evolution; this reflects in the design of the signal plan and its offerings.

3.2.1 Political context

In the mid-90s the European Union started considering developing its own satellite navigation system, with three main motivations: to increase control on satellite-based safety-critical navigation systems, to ensure a positioning service independent of the risk of potential U.S. policy changes to GPS availability, and to support EU industry competitivenes in the global market of satellite navigation.

Once defined the political scope of the project, parallel efforts were launched to define service levels, space and ground architectures, frequency plan and signal design. At the same time, a strong formal cooperation between EU and U.S. resulted in the release of a framework to achieve full interoperability and radio frequency compatibility between Galileo and GPS.

3.2.2 Service offering

Similarly to GPS, Galileo's signal plan is intended to offer open civil access as well as access restricted to authorized users. Differently than GPS though, the service plan for Galileo includes from the start a broader range of options:

- **Open Service** provides positioning and timing information worldwide, and is meant to be accessible free of charge from any user equpped with a Galileo compatible navigation receiver. This is similar to the service level of GPS L1 C/A, L2C or L5

- **Public Regulated Service** provides the same capabilities as the Open Service signal, but it's meant to be restricted to only government-authorized users, for use in sensitive applications. This is similar to the service level of GPS P(Y) or M codes

- **Commercial Service** is included in the current service specification but its definition is as of now undergoing. The idea for the CS is to offer "added value" with respect to the Open Service, mainly in terms of high accuracy and authentication. In addition, this service is going to provide the possibility of globally transmitting external data in real-time. This service offering is completely new in the GNSS landscape and has currently no similar counterpart in other constellations

- **Search and Rescue** is meant to enhance the current capability of another system, namely LEOSAR, for use in assistance of users in distress. Through SAR users that need assistance can transmit a beacon signal that can be received by Galileo satellites and then relayed to ground stations that can then proceed with locating the source of the beacon by reversing the PVT equation normally used in GNSS

- **Safety of Life** is a service aimed at offering worldwide system integrity. It has been foreseen by the Galileo program but its actual implementation has been postponed to later phases of the project

Each of these services is going to be available on one or more frequencies of Galileo's signal plan, which is described next.

3.2.3 Signal plan

The Galileo signal is in general very similar to that of GPS, in the sense that in most general terms it's composed of the sum of a PRN spreading code and a navigation message. Some peculiar characteristics set the Galileo signals apart from the GPS ones.

The main difference is that the PRN codes employed in Galileo are not pure Gold codes as in the case of GPS, but have been developed with genetic algorithms starting from such standard codes. Each spreading code is generated by the sum of a primary and a secondary code. Some of the primary codes can be generated using LFSR, but most of them need to be stored in memory. The secondary codes are 2 to 3 orders of magnitude shorter than the primary codes, and can be

easily stored in a few memory words. The reason for this tradeoff is that Galileo codes are heavily optimized for their cross-correlation properties and interference mitigation capabilities.

On the other hand the navigation message is of three different kinds, depending on the band of transmission and on the service level. For OS, two types of message are available: F/NAV and I/NAV. For CS, I/NAV and C/NAV, the C/NAV being encrypted and not covered within the public ICD. The main difference between these messages is the length of the transmitted pages, which is bound to the chip rate of the signal they're transmitted on. In general, though, each of them conforms to a general navigation message structure. Independently from its length, the navigation message is composed of a repetition of frames which share the same structure. Each frame is composed of a number of sub-frames, which are in turn composed of a number of pages. Pages can be of different type, which is identified in a dedicated field. Each page type contains different sets of information. In general though, Galileo navigation message contains three main types of information:

- Ephemeris: characterize the orbit of each satellite, allowing users to calculate its coordinates in a Earth Centered, Earth Fixed (ECEF) reference frame
- Time information: such as GST, clock correction, GST-UTC and GST-GPS conversion parameters
- Ionospheric correction: a set of parameters that allow receivers to adjust the PVT equation to account for ionospheric interference

Galileo signals are transmitted over four bands within the official RNSS and ARNS frequency bands. These four bands are named E1, E5a, E5b and E6. The center frequency of these bands ranges from 1176.450 MHz to 1575.420 MHz, while the transmitted bandwidth ranges between 20.460 MHz and 51.150 MHz. A full account of the frequency allocation and bandwidth distribution is provided in figure 3.1 and table 3.1.

Signal	Carrier Frequency (MHz)
E1	1575.420

Signal	Carrier Frequency (MHz)
E6	1278.750
E5	1191.795
E5a	1176.450
E5b	1207.140

Table 3.1: Galileo signal plan

All Galileo signals are RHCP, and independently from the modulation are composed of in-phase and quadrature components. In-phase components carry the navigation message, while quadrature components only carry the PRN code with no data; such pilot signal has the purpose of aiding initial signal acquisition.

As in the case of GPS, usage of PRN allows for transmission of multiple signals from different SVs over the same frequency - an implementation of Direct Sequence Spread Spectrum Code Division Multiple Access (DS-SS CDMA). Apart from that, Galileo employs a Binary Offset Carrier (BOC) modulation to allow for interoperation with the GPS signal, as for example in the case of L1/E1 the same frequency band is used by the two system. The topic of interoperability is expanded in a following section.

Looking at the signal plan, one might notice that Galileo transmits a variety of signals much larger than the current GPS one. This variety is characterized by variations in the code length and navigation message data rate. This is an engeenering design choice to support a wide variety of use cases, such as users that receive a very faint signal, or users that move at high speed.

3.2.4 System architecture

Transmission and control of the described range of signals is performed within Galileo's system infrastructure, which is composed of three major blocks: the Core Infrastructure, Service Facilities and Support Facilities.

Galileo's Service Facilities provide steering corrections to the CI by monitoring the alignment of time and reference geodetic model with the international meteorological standard (UTC and ITRF, respectively).

Galileo's Support Facilities are not directly involved in the routine provision of

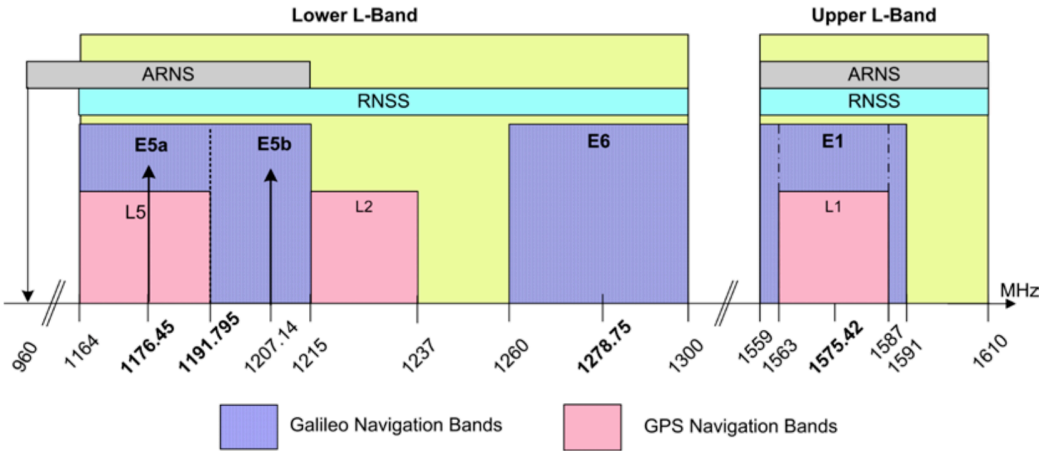

Figure 3.1: Galileo and GPS frequency spectrum

services but play a major role in the deployment, testing and subsequent control of satellites into orbit.

Galileo's CI is instead the main part of the system infrastructure and can be subsequently divided in three main segments: Galileo's Space Segment, Ground Control Segment and Ground Mission Segment.

The Space Segment, as defined in the reference constellation, is composed of 24 operative satellites distributed in 3 circular orbits with a radius of 29,600 km and an orbital period of approximately 14h. At the time of writing, information on the actual Galileo constellation can be found in [1].

Galileo's satellites are currently divided in four families: Giove-A and Giove-B are the initial prototypes and have been used for the initial testing phase; their designed lifetime is of 27 months. Currently being deployed is the IOV family, with improved design and a projected lifetime of 12 years; these satellites are used for the In-Orbit Validation phase of the project. Similar to the IOV satellites are those in the last family, namely FOC; these are meant to support Galileo's Full Operational Capability and has been deployed starting from Q3 2014. In terms of functionalities, overall budget envelope and performances these last two families are close to each other.

The constellation is managed during normal system operation by the GCS, which is composed of two kind of facilities: control centers (GCC) and telemetry stations (TT&C).

The GCCs perform duties related to monitoring and control of the constellation, together with planning, flight dynamics and operations preparation of the same. Through TT&C, instead, ground control performs data exchange with the entire constellation through 11m antenna dishes that operate in the S-band. Normally these stations are used to perform tele-command of the satellites and to receive telemetry data that is further used for off-line satellite orbit determination.

The signal sent to the user by spacecrafts is generated by the GMS, a network of sensor stations, control centers and up-link stations that interoperate to perform a distinct set of operations:

- generation and distribution of Galileo System Time (GST) to satellites and ground segments
- generation and distribution of Galileo OS and PRS navigation messages
- distribution of mission data provided by Galileo Service Facilities for CS and SAR signals
- provision of the physical interface with the US Naval Observatory for generation of the GPS Galileo interoperability products like the GPS to Galileo time offset

3.3 Other constellations

In the GNSS panorama, there are two other constellations worth of note for the role they're playing or will predictbly play in shaping up the future of a common GNSS infrastructure: Russian GLONASS and Chinese BeiDou.

3.3.1 GLONASS

With a history similar to that of GPS, the GLONASS program was initiated in the 1970s to support military requirements. Later on, early tests demonstrated the possibility of handling also civilian use. The first launch dates back to 1982. Despite having caused interference within the radio astronomy observation band, development of the program continued steadily - even past the demise of the Soviet Union. In 1995 the last batch of satellites was launched to complete the

constellation, which nominally consisted of 24 satellites. Shortly after, however, a number of older satellites failed and the Russians failed to replenish the constellation. In 2001, restoration of the degraded constellation was made a top government priority; since 2011 GLONASS is again fully operational with global availability.

GLONASS constellation consists of 24 satellites, 21 of which fully operative and 3 acting as spares. The satellites are organized in 3 orbital planes 120 apart from each other in right ascension with a period of 11 hours and 15 minutes.

Such configuration provides continuous visibility of no less than 4 satellites to over 97% of Earth's surface. A full 24 satellite configuration would instead provide continuous visibility of no less than 5 satellites to more than 99% of the Earth's surface.

Differently than other systems, GLONASS historically used FDMA as a multiplexing technique, which increases complexity on the receiver side but allows for better interference rejection properties. As of 2016, a new ICD was published containing new CDMA signals, namely three open (L1OC, L2OC and L3OC) and two restricted (L1SC and L2SC).

The older FDMA signals are transmitted on a different frequency by each satellite in the L1 and L2 bands, following the equation

$$\begin{aligned} f_{K1} &= f_{01} + K\Delta f_1, \\ f_{K2} &= f_{02} + K\Delta f_2 \end{aligned} \tag{3.1}$$

where

$$\begin{aligned} f_{01} &= 1602MHz, \Delta f_1 = 562.5kHz \\ f_{02} &= 1246MHz, \Delta f_2 = 437.5KHz \end{aligned} \tag{3.2}$$

To avoid interference with the radio astronomy band, the value of K changes between satellites in a range that goes from -7 to 4. To account for just 12 different available frequencies, antipodal satellites are going to transmit on the same frequency (i.e. will share the same value of K). Since these satellite pairs are on opposite sides of Earth's surface, users are not impacted.

GLONASS signal is of two types: C/A code or P code. The latter is restricted to military use, while the former is open for civilian use. Both signal types transmit a navigation message with a rate of 50 bps. Also, both signals are composed by a PRN code message and a navigation message. The main content of the navigation message is represented by the ephemeris, but in addition to that GLONASS transmits other data such as epoch timing, synchronization bits, satellite health and age of data fields. Each transmission contains accurate ephemeris information for the transmitting satellite and approximate (i.e. almanac) information about all the satellites in the constellation.

3.3.2 BeiDou

Another global player in radio navigation satellite systems comes from China under the name of BeiDou. The initial concept was developed in the 1980s, and the first launch of an experimental satellite happened in the year 2000. In 2007, the first version of BeiDou (BeiDou-1) became fully operative with a regional coverage of China and the surrounding countries.

As of mid-2007, deployment of the BeiDou-2 system began with an expected timeline lasting till 2020, which will bring BeiDou-2 at full operativity and global coverage. This will consist of 35 satellites, organized in different group. 5 satellites will be on geostationary orbit; 3 of them will be on inclined geosynchronous orbit; finally, 27 of them will be displaced in 3 medium Earth orbits.

BeiDou-2 will transmit on 5 bands: E1, E2, E5B and E6. The system will transmit two types of signal: a free service for civilian use and a restricted one for military and government use. The free service is claimed to have location accuracy of 10m, time accuracy of 10ns and velocity accuracy of 0.2m/s. The military service instead will have a location accuracy of 0.1m. The modulation of these signals uses CDMA techniques similar to that of Galileo; the chipping rate and the data rate are also comparable if not identical to that of Galileo OS.

Although little is known about the actual structure of the system and the service levels to be offered, it's clear that the Chinese government is aiming at becoming more than competitive in the arena of radio navigation systems. The speed at which the Chinese deployed satellites in the past years hints at the idea that BeiDou-2 will be soon an equal alternative to GPS and Galileo.

3.4 Interoperability challenges

As the number of global systems increases, and as the radio spectrum gets more and more crowded, the problem of interoperability (or, put in different terms, that of interference) becomes more and more relevant. In general, a choice has to be made: should system be independent and completely isolate, or should their design be aware of the presence of other navigation systems?

The answer came during the third meeting of of the Providers' Forum of the International Committee on Global Navigation Satellite Systems (ICG) in 2008. Here it was agreed that at a minimum all GNSS signals and services should be compatible. At the same time, to the maximum extent possible open signals and services should also be interoperable, in order to maximize benefit to all GNSS users.

In the same committee, a contextual definition of the two terms was provided:

- **Interoperability** refers to the capability for all the systems to be used together to provide better capabilities at the user level than each individual system, at a minimal additional receiver cost or complexity
- **Compatibility** means that two systems can be used individually or together with no unacceptable interference or other harm to the other system. For this the ITU provides a framework for discussions on radiofrequency compatibility.

This formal agreement resulted in a number of significant decision in the evolution of the design of all major GNSS.

For example, when designing the frequency plan of Galileo, one of the drivers was the selection of common center frequencies for some of the signals: this was to prevent unnecessary increase in the cost of multi-frequency receivers and to make combined processing of phase observations possible.

From the point of view of signal spreading modulation, the choice of CDMA was a significant step in the design of interoperable systems. On the same topic, while the initial definition of GLONASS used FDMA, new signals are being included that use CDMA.

At the data level, two other notable examples are the choice of the coordinate reference frame and that of time reference. In particular, the Galileo Terrestrial

Reference Frame and the GPS coordinate frame (WGS84) differ a few centimeters from the International Terrestrial Reference Frame, hence making the two interoperable for most applications. In the case of time, both GST and GPS time are different real-time realizations of UTC, and the service providers have agreed to broadcast the time offset between the two standards.

The question of interoperability and compatibility has had repercussions also on the organizational and political level. Although later unsatisfied by the role in the project, China signed in 2004 an agreement on the cooperation in the Galileo program between the "Galileo Joint Undertaking" and the "National Remote Sensing of China", which resulted in 11 cooperations projects being signed within the program between China and EU in 2006.

4

Data authentication in Galileo

So far we've described the classic radio satellite navigation problem and solution. This generally accepted model works well but provides no security guarantees. In particular, it doesn't prevent malicious users to forge ad-hoc signals and impersonate a satellite, or even a whole GNSS.

In this chapter we'll describe the general data authentication problem for open signals in GNSS environments, the threat model and the possible attacks. We will then move on to describe TESLA (Timed Efficient Stream Loss-tolerant Authentication), a data authentication protocol suited for streaming, lossy channels that the Galileo program chose to adopt in order to secure the navigation message. We'll also describe the proposed implementation of TESLA in the current Galileo subframe structure.

4.1 TESLA

Certain characteristics of the Galileo signal make the problem of authenticating the data very peculiar. Among others:

- communication is unidirectional, from satellite to receiver(s)
- receivers can connect at any point in time
- the channel is lossy and doesn't guarantee data delivery

- communication happen on a very low-speed channel

The first three traits make it easier to associate the Galileo signal to a multicast streaming communication. For this kind of communication, two schemes have been proposed in [2]. One of them, TESLA, offers source authentication and integrity protection of the message with a minimal overhead, strong loss resistance and high scalability, at the price of delaying the verification process with respect to the message reception time. The second of them, EMSS, adds non-repudiation on top of those features. The European Commission chose to implement TESLA for authenticating the Galileo Open Service navigation data [3], and therefore we shall now put our focus on it.

As an overview, TESLA builds on top of classic asymmetric authentication schemes and MAC for data authentication. What TESLA does is to authenticate the data it sends and to disclose the key that authenticated that data in a later packet. Moreover, keys are connected in a sequence by using a key generation function that takes a previous key as input. To account for different reception speeds, TESLA supports using multiple chains, each with a different validity time for a single key.

In our overview of the algorithm, we'll first look at how a single key chain works, and then extend the concept to multiple chains.

We start from a stream of messages M_j, each one sent after the other and received in the same sequence. For each of these messages, TESLA computes a MAC using a key K_j and attaches it to the message. Moreover, TESLA attaches to the same message the key $K_{j-\delta}$ it used to authenticate a previous message $M_{j-\delta}$, where δ is a parameter called *disclosure lag*, chosen so that it's guaranteed that every receiver received $M_{j-\delta}$ by the time $K_{j-\delta}$ and M_j are sent out. This is referred as the *security condition* of the TESLA protocol, and it means that there's no chance for an attacker to read M_j, forge or modify $M_{j-\delta}$ and have the latter accepted by a receiver.

Since keys are disclosed in clear, TESLA provides a way to verify that the key is authentic by using other keys in the chain, plus a bootstrap mechanism. The sender start by generating a random key K_l that remains secret and that's never used to sign any message. After this, the sender computes a chain of keys by successively applying a one-way function F to the keys in the chain:

$$K_i = F(K_{i+1})$$

After l steps, the chain is complete and the last application of F yields what's called a *root key*:

$$K_0 = F^l(K_l)$$

At this point the keys are broadcasted in reverse order than they're computed, i.e. K_0 is sent before K_1, etc. This makes so that it's computationally infeasible for an attacker to calculate K_{i+1} given K_i since F is a cryptographically secure one-way function. At the same time, once K_{i+1} is received it's easy for the receiver to verify that the key K_i is authentic, as it needs to only apply F to K_{i+1} once and verify that the result is K_i. The only requirement is that the root key is signed using an asymmetric scheme during a bootstrap phase, so that receivers can verify that the chain has been generated by the legitimate sender.

Another important aspect of TESLA is the use of the same key across several messaeges. In fact, the validity of a key is not tied to a number of messages but rather to an amount of time. This way the underlying streaming protocol can have a variable streaming rate without increasing the overhead on the receiver's side.

To give support to different types of receivers with different network access speed, TESLA suggests the implementation of multiple simultaneous chains with different disclosure periods δ (this particular aspect is not considered in OSNMA since the network bandwidth is fixed and the same for all receivers).

One important precondition that needs to be satisfied for the security condition to hold is that sender and receivers have clocks that are loosely synchronized within a disclosure period (otherwise the security condition cannot be practically verified).

4.2 Galileo OSNMA

In 2016, a specification for a data authentication protocol in Galileo has been released under the name of OSNMA [3]. This specification builds on top of the TESLA algorithm just described with a few notable exceptions. We'll give here

an introduction to this protocol to contextualize the analysis that follows.

The OSNMA specification describes two distinct sets of information that are sent along together but at different rates: the HKROOT and the MACK sections. The HKROOT contains a set of headers needed to correctly interpret the authentication data, the root key of the chain in force and its digital signature. The MACK section contains a configurable number of MAC which authenticate different groups of fields contained in the navigation message, and the key with which the MAC are generated.

These two pieces of information are transmitted together at a rate of 40 bits per page (2s) in the field called "Reserved 1" in the Galileo ICD [4]. The HKROOT section occupies 8 of those bits, while the remaining 32 are allocated for the MACK section.

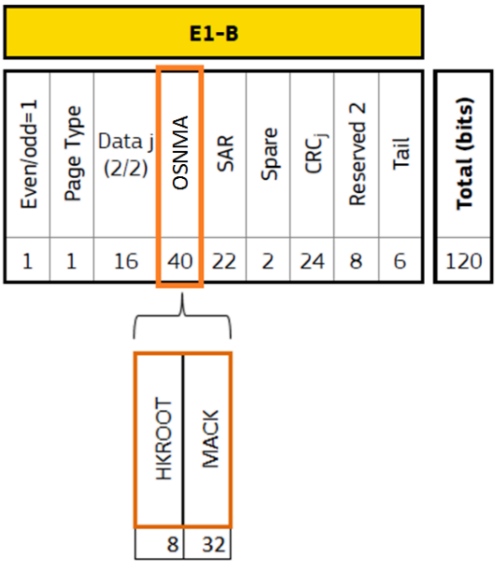

Figure 4.1: OSNMA message structure in Galileo E1B I/NAV frame

In total, a HKROOT section has a variable length and it's transmitted in blocks of 104 bits - one block per subframe. The MACK section instead has a fixed length of 480 bits and can contain a variable number of MACs and keys (up to 480 bits; the remaining bits are padded).

The HKROOT can also be thought as composed of two sections: the header and the DSM-KROOT. The header contains information about the overall NMA status and about the DSM (Digital Signature Message) transmitted in the section. The first header is composed of three fields: the operational status of NMA, the status of the DSM and the ID of the chain in force. The second header contains the ID of the DSM, which identifies also if the transmitted information is related to a root key being authenticated, or if it's a message related to a key revocation. Also, an indication of the current block being received is sent in this header. The DSM-KROOT instead contains a set of parameters like the total number of blocks, the hash function to be used and the key size, together with the DSM and the associated key. The DSM-KROOT can optionally be swapped with the DSM-PKR section for key revocation.

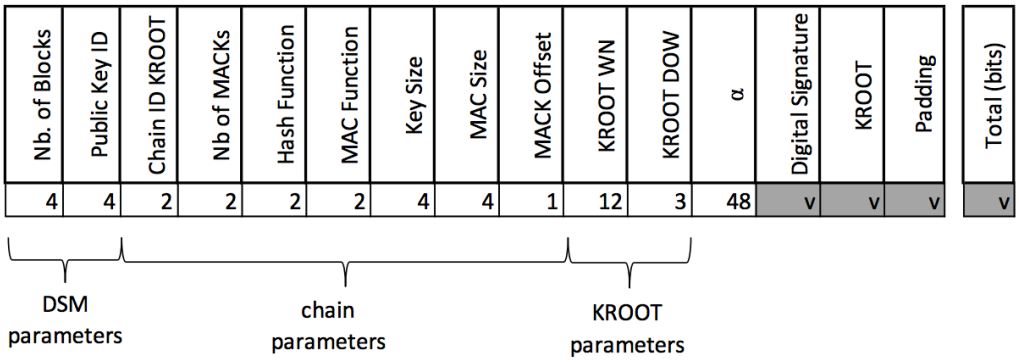

Figure 4.2: Fields in DSM-KROOT section. "v" in the bit size stands for "variable"

The MACK section instead contains the authentication codes for the data transmitted in the navigation message. This data is authenticated with different MAC, therefore the MACK message also contains information on what data is included in the message from which the MAC has been generated, in the form of a 4 bit field called ADKD (Authentication Data & Key Delay). Together with this, the PRN of the satellite being authenticated and an Issue of Data field indicating if the data transmitted is new are sent.

After the MACs, the key that generated them follows, the size of which is specified in the DSM.

The pieces of information included in OSNMA are to be used by the receiver

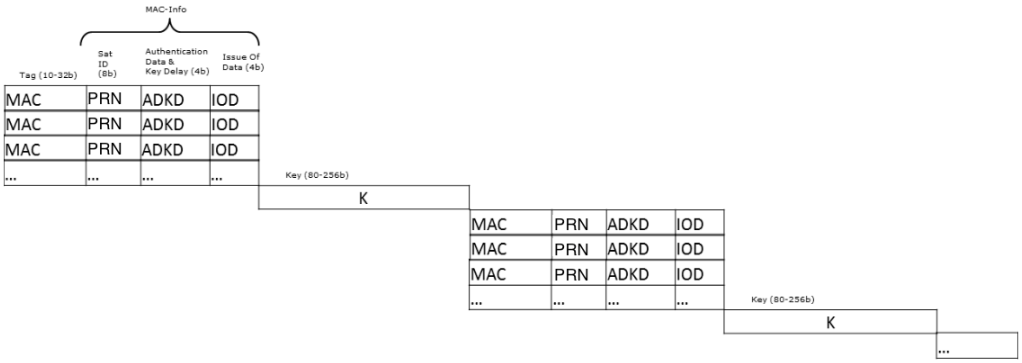

Figure 4.3: MACK section structure

as follows:

- the receiver must first receive and decode in full the root key K_R transmitted in the DSM
- the receiver computes the DSM for the root key as follows

  $$M = (\text{NMA\_Header}\|\text{CIDKR}\|\text{NMACK}\|\text{HF}\|\text{MF}\|\text{KS}\|\text{MS}\|\text{MO}\| \text{KROOT WN}\|\text{KROOT DOW}\|\alpha\|K_R)$$

  $$S = signature(M, K_{pub})$$

  where for $signature()$ ECDSA is proposed in [3] with several supported signature lengths, the actual value of it is univocally identified by the *Public Key ID* transmitted in the DSM header. K_{pub} is the long term public key identified by the above mentioned ID and stored in the receiver's memory. All the other parameters are defined in [3].

  If the computed signature matches the received DSM, then the received KROOT is to be considered valid
- once the root key has been verified, the receiver can start to read a MACK section, and then decode the MACs and the key K_j associated to them. Depending on how the protocol is configured, the receiver might have to decode the next subframe to get the key that generated the MACs in its possession.
- the receiver identifies the distance d between the root key K_R and the key sent in the MACK section K_j

- the receiver verifies the key sent in the MACK section against the root key already verified by applying d times the key generating function F on K_j:

$$K'_R = F^d(K_j)$$

- the receiver verifies that $K'_R = K_R$. If the equality holds true, then the received key K_j is validated and the receiver can proceed in verifying the actual data in the navigation message against the MACs already received.

- the navigation data is verified by building a message in the format

$$m = (\text{PRN\_N} \| \text{PRN\_A} \| GST_{SF} \| \text{CTR} \| \text{NMA\_Status\_Header} \| \text{navdata})$$

  where PRN_N is the PRN of the satellite being authenticated, PRN_A is the PRN of the satellite sending the authentication data, GST_{SF} is the Galileo system time of the start of the subframe in which the MAC is contained, CTR is the position of the MAC in the MACK section, NMA_Status_Header is the NMA status field in the NMA header, and navdata is fetched looking into the ADKD field of the MAC-Info section of the MAC and combining the data field according to [3]. The message thus constructed is then verifying by applying to it the hash function specified by the protocol together with the key K_j. If the resulting hash, truncated accordingly to the MAC length specified in the protocol configuration, matches the MAC in possession of the receiver, then the data is considered authenticated and valid.

One important aspect of OSNMA is the generation of the key chain. According to the protocol, the ground segment generates one chain that's valid for all satellites. This means that when two subsequent keys K_i and K_j are received from a same satellite, they're not at one step distance (i.e. $K_i = F(K_j)$ doesn't hold) but in between them there are other 35 keys, one for each satellite (where the total number of satellites is nominally 36 but should be configurable according to the specification).

5

Receiving OSNMA: First Steps

In the next two chapters we'll take a critical look at the OSNMA specification, trying to reformulate it from a receiver point of view. The goal is to identify key areas of possible improvement for the protocol, together with critical aspects for a receiver implementation, and to provide suggestions for both. We'll start with analysing what it takes for a new receiver to be able to authenticate the navigation message.

When thinking about a Galileo receiver capable of supporting OSNMA, we can identify two distinct modes of operations: a bootstrap phase and a nominal one. In the bootstrap phase the receiver doesn't possess any data apart from a public key capable of authenticating a DSM; therefore in this phase the receiver is not formally capable of authenticating the navigation message, and needs to perform operations which can bring it into the nominal mode. In this phase the receiver is capable of authenticating the navigation message using the data it has stored locally and the MACK sections it receives together with the unauthenticated message.

This distinction is purely analytical in nature, but can help breaking down the problem into distinct blocks, which will be separately analyzed in the chapters to follow.

For the bootstrap phase we'll consider a generic Galileo receiver with no access to any data network other than the radio channels over which the Galileo satellites transmit. This receiver has never received any navigation data before, and only contains onboard a set of public keys that should allow it to verify the authenticity of the root keys sent in the DSM-KROOT section.

To keep the discussion focused, we'll skip over the specific details of acquiring one or more signals and we'll assume the receiver can enter the acquisition phase for a sufficient number of in-view satellites, and to successfully move into the tracking phase which will allow it to decode the Galileo navigation data frame.

Under these conditions, when the receiver starts its operations it has no knowledge about the key chain currently in force, and therefore if it wants to authenticate the navigation message the first thing it has to do is to receive a DSM-KROOT and decode the root key. We'll start our analysis from here.

5.1 Root key authentication

The root key is sent in the HKROOT section of the navigation data frame. This section of data is transmitted at a rate of 8 bits every 2s (i.e. 4bps) and is composed of two headers of 8 bits each, and of a variable number of blocks of 104 bits. Each subframe contains the two headers and a DSM block. According to the current specification, the number of blocks that compose a DSM can vary between 6 and 16 (but there are 5 reserved values so it could possibly be longer). In order for a receiver to be able to correctly compose a DSM, in the first block the total number of blocks is sent.

5.1.1 HKROOT receiving time

The first simple conclusion we draw from what we've seen so far is that reception of a whole DSM takes between 180s and 480s. In order to speed up the bootstrap operation a receiver might cache the navigation message sent, provided it won't use it until it has been authenticated in full. The amount of caching memory needed will be analyzed later on in the chapter.

5.1.2 Variable size of the DSM

A second observation is about not knowing the precise size of the DSM upfront. Since the receiver needs to cache the DSM until it's fully received and can be decoded, two general approaches are possible:

1. a fixed amount of memory is allocated that accounts for the longest possible DSM. This memory could be separate from the RAM associated with the CPU that perform the authentication operations, in order to avoid security problems due to buffer overflows. This could be expensive and inefficient (for example, if a DSM is only 6 blocks long, then the memory reserved for the other 10 blocks is wasted)
2. the RAM of the computational unit is used to cache the DSM until it's possible to decode it. In this case, a variable amount of memory is allocated, and two strategies could be followed to manage the length variability:
   (a) the receiver makes sure that it starts decoding a DSM from its first block, which contains the information about the total number of blocks. This means potentially discarding several minutes of data
   (b) the receiver stores the blocks as individual pieces of information and then recombines them at the end (for example, using some form of sorting algorithm), or on the fly (for example, using something like a linked list)

This scenario offers the possibility for several considerations. The first one is that, no matter the strategy chosen to store the DSM blocks, care must be taken not to expose the receiver to memory overflow attacks. Since the data that's being received at this point in time is not authenticated yet, it could also be forged by an attacker who would send more blocks than what stated in the first DSM block. If a receiver doesn't implement a preventive check, this kind of attack might allow an attacker to overwrite some reserved memory with executable malicious code. A simple check that the number of the block being processed is within the advertised length is sufficient to prevent this kind of vulnerability.

The second consideration is that neither of the scenarios in option 2) are ideal. In fact option 1) is safer, but results in even longer bootstrap times. The second option is faster but it's based on the assumption that the satellites will keep

sending the same key over and over again (i.e. if blocks with lower indexes are missed, then they can be received later on by a subsequent transmission of the same key). Moreover, dealing with the unknown length of the DSM requires more expensive data structures and operations.

That said, an ideal approach for a receiver would be to learn the total length of the DSM no matter at what point of its transmission it starts to receive. This way the receiver could allocate all the needed memory at once, and subsequently fill in the gaps with the received blocks (provided it also implements the check to prevent memory overflow attacks). In the OSNMA protocol this could be achieved by sending the field *Number of blocks* in the DSM header rather than the first block of the DSM. This would increase the size of the DSM Header from 8 bits to 12 bits, resulting in an additional overhead that ranges between 24 bit (when a DSM is composed of 6 blocks) to 64 bits (when a DSM is composed of 16 blocks).

5.2 Initial key authentication

The specification provided in [3] recommends for a single key chain an extension of 2^{25} to 2^{26} keys. The actual duration of such a chain is dependent on the key size, the number of satellites and the number of MACK sections per subframe; with a key size of 82 bits, 36 satellites and 3 MACK sections per subframe, the chain would have a duration of about 4 months.

Under these assumptions, we can try to analyze the scenario in which a receiver with no prior navigation data starts to receive a navigation message inclusive of DSM and verification tags, and subsequently tries to authenticate the received key against the root key of the chain.

One hypothesis we make to analyze this scenario is that the key included in the DSM-KROOT section is the key with index 0. That is, the constellation doesn't authenticate any other, more recent, key. This clarification is necessary since the protocol specification declares the possibility of transmitting more recent keys (i.e. with index higher than 0) in the DSM, but doesn't provide any detail on how that would work. To separate the two problems, we provide here a worst-case analysis that's independent from the transmission of floating KROOTs.

5.2.1 Receiver operation

Having laid out the context, we can proceed to describe the set of operations a receiver is required to perform in order to authenticate the navigation message.

Let's assume the receiver already received and decoded a sufficient amount of data to calculate the pseudorange against a satellite and to authenticate mentioned data. This includes having at hand at least one MAC t_m, the key that has been used to produce it K_m, and the root key K_0.

Given only this set of data, in order to authenticate the key K_m the receiver must perform precisely m invocations of the one-way hashing function F in order to verify that $K_0 = F^m(K_m)$. At the end of the calculation, the receiver will obtain $K'_0 = F^m(K_m)$; if $K'_0 = K_0$ (i.e. the calculated key matches the one embedded in the DSM), then the authentication of K_m is successful. Otherwise, the key and associated data must be discarded.

We can then easily see that if a receiver starts to receive data at the end of a key chain of length $L + 1$, the initial key authentication will require applying L times the hash function to K_L.

5.2.2 Initial authentication benchmarking

Following this result, we perform a benchmark to understand time and resource consumption to aid hardware design and implementation for receivers that should support OSNMA.

The algorithm described so far contains three degrees of freedom that have been analyzed separately: chain length, key length and selected hash function. As stated, the recommended chain length is between 2^{25} and 2^{26}; to provide a more extensive overview our benchmark ranges between 2^{20} and 2^{30} keys. The options for key length are fixed and provided in the *Key Size* included in the DSM-KROOT; these range between 80 and 256. The possible hash functions are SHA256, SHA3-224 and SHA3-256.

Given the limited availability of open source libraries that support SHA3 at the time of writing, we've chosen to perform the benchmarking using **Python 3.6**. Since this dynamic language is not what usually gets used to write embedded code, we've compared its performances with a C implementation of the same logic. The two programs both measure the time it takes to perform 2^{20} computations

of a SHA256 chain.

The first test measures how much time it takes to compute the whole chain of keys, for different lengths of the chain. Tested lengths vary between 2^{20} and 2^{30} keys, so that the suggested values of 2^{25} and 2^{26} lie in the middle of the range. The computation is done against a fixed key of 80 bits of length, and the same test has been performed for all the hash functions supported by the protocol.

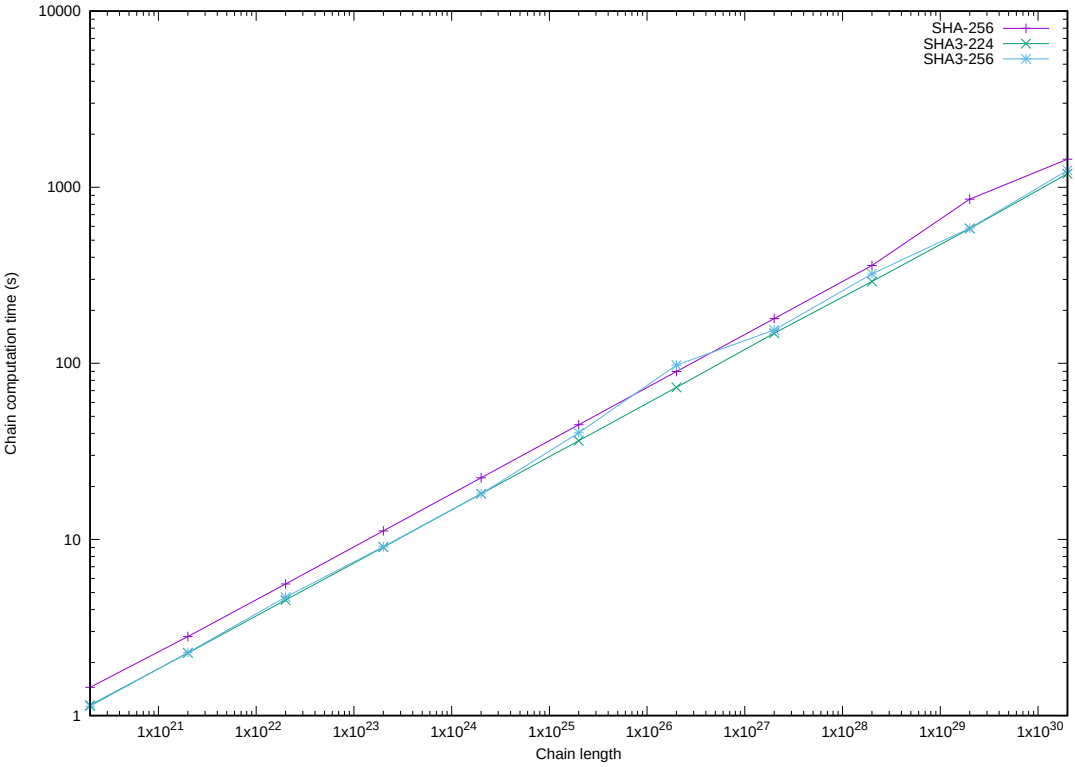

Figure 5.1: Chaing length vs computation time of the whole chain

The results of this first benchmark can be seen in figure 5.1. The values that make up the graph can be seen in table 5.1. It's important to observe that the both axis are in logarithmic scale. We can observe that the relationship is basically linear with little difference between hash functions: that is, no matter the function chosen, doubling the size of the chain directly doubles the time it takes to compute the chain.

Chain size	SHA-256	SHA3-224	SHA3-256
2^{20}	1.4449	1.1455	1.1296
2^{21}	2.8077	2.2665	2.2768
2^{22}	5.5871	4.5320	4.7067
2^{23}	11.189	9.0380	9.0916
2^{24}	22.408	18.161	18.170
2^{25}	44.857	36.362	40.316
2^{26}	89.838	73.052	97.746
2^{27}	179.45	148.20	154.51
2^{28}	359.15	291.50	321.99
2^{29}	855.27	582.81	586.11
2^{30}	1442.6	1192.6	1247.0

Table 5.1: Key chain computation time [s] vs key chain length

Another general observation is that the computation time improves slightly for the SHA3 family of hash functions, but in general the variability in computation time is dominated by the chain length.

One important aspect to focus our attention on is the relative significance of this benchmark with respect to the average computation power of a common receiver. The numbers just described have been obtained by running the code on a MacBook Pro sporting a 2.6GHz Intel Core i5 processor. A benchmark sheet such as [5] can show that such a processor can handle roughly 50000 DMIPS. As a reference, the STA8088EXG GNSS receiver [6] sports an ARM9 processor clocking at a maximum of 208MHz. Usually in such receivers the CPU clocks at much lower rates than their maximum (see for example [7]). According to the ARM Information Center website [8], the maximum performance of the ARM9 family reaches 1.1 DMIPS per MHz, which means we can assume the reference CPU can achieve roughly 230 DMIPS when at full speed.

In order to translate these observations to an estimate of the running time for the chain computation on an embedded device, we've correlated the running time of the Python code to that of its C counterpart. Surprisingly, the same benchmark for a key chain of length 2^{20} over the SHA-256 function seems to perform better in Python than in C, as exposed in table 5.2. As a reference, the Python

snippet is provided in listing 5.1 and its C counterpart in listing 5.2. This is because the Python implementation uses the same libraries as the C code, but the Python environment calls out to architecture-specific assembly implementations of those functions. In any case, this shows that the performances of the two implementations are within the same order of magnitude, so comparing directly against the Python results is a good approximation of the best case scenario.

Listing 5.1: Computing 2^{20} SHA256 hashes in Python 3.6

```
import hashlib
import time

iterations = 10
chain_lengths = [103680, 414720]#range(20,30+1)

for exp in chain_lengths:
    chain_length = exp #2**exp
    total = 0
    for _ in range(0, iterations):
        start = time.time()
        for _ in range(0, chain_length):
            hashlib.sha256(b"1234567890").hexdigest()
        end = time.time()
        total += end - start
    print(exp, total/iterations)
```

Listing 5.2: Computing 2^{20} SHA256 hashes in C

```
#include <stdio.h>
#include <openssl/sha.h>
#include <sys/time.h>
#include <sys/resource.h>

double get_time()
{
    struct timespec t;
    clock_gettime(CLOCK_PROCESS_CPUTIME_ID, &t);
```

```
    return t.tv_sec + t.tv_nsec*1e-9;
}

int main() {
  double avg = 0.0;

  unsigned char hash[SHA256_DIGEST_LENGTH];
  SHA256_CTX sha256;

  for (int j = 0; j < 10; j++) {
    double start = get_time();

    for (int k = 0; k < 10248576 << 2; k++) {
      SHA256_Init(&sha256);
      SHA256_Update(&sha256, "1234567890", 10);
      SHA256_Final(hash, &sha256);
    }

    double end = get_time();
    avg += end - start;
  }

  printf("avg: %f\n", avg/10);

  return 0;
}
```

Python	C
1.4449s	2.2960s

Table 5.2: Average computation time for the calculation of 2^{20} SHA256 hashes in Python vs C

In other words, this comparison tells us that in order to get an estimate of how long it would take to compute the whole key chain on an embedded CPU

of a Galileo receiver we can simply relate the running times t_j just measured, the processing power of the CPU where the benchmark was run P_{BM} and the projected processing power of the CPU of the receiver P_R. The processing power is measured in DMIPS, a definition of which can be found in [9]. A simple proportion shows that

$$t_j/P_{BM} = t_{rj}/P_R \tag{5.1}$$

where t_{rj} is the estimate of the running time on the receiver processor. This equation solved for this last variable yields:

$$t_{rj} = t_j \frac{P_{BM}}{P_R}$$

Substituting the variables with the numbers devised above we can obtain an approximate conversion as follows:

$$t_{rj} = \frac{50 \cdot 10^3}{230} t_j \ = 246.30 t_j \tag{5.2}$$

Applying this conversion rate to table 5.1 we obtain the numbers in table 5.3.

Chain size	SHA-256	SHA3-224	SHA3-256
2^{20}	355.88	282.14	278.22
2^{21}	691.54	558.24	560.78
2^{22}	1376.1	1116.2	1159.3
2^{23}	2755.9	2226.1	2239.3
2^{24}	5519.1	4473.1	4475.3
2^{25}	1.1048e4	8956.0	9929.8
2^{26}	2.2127e4	1.7793e4	2.4075e4
2^{27}	4.4199e4	3.6502e4	3.8056e4
2^{28}	8.8459e4	7.1796e4	7.9306e4
2^{29}	2.1065e5	1.4355e5	1.4436e5
2^{30}	3.5531e5	2.9374e5	3.0714e5

Table 5.3: Approximate key chain computation time [s] vs key chain length for an average GNSS receiver

By looking at this result we can see how the worst case scenario might result in severe energy consumption by the receiver itself. Considering just the lengths that have been proposed in the specifications, we can see how a receiver might spend between 3.5 and 5 hours of computation just to perform the authentication of a single key. Considering the case of mobile devices such as smartphones, this is an amount of time that can have a noticeable negative impact on battery level and might make data authentication infeasible for common uses.

To complete the analysis, we evaluated the change in computation time as the key length varies. As figure 5.2 shows, there's a slight increase in computation time of a single hash in the case of SHA-256 when the key size reaches around 180 bits, but in the case of the other functions the computation time remains constant. This leads to the conclusion that computation time is bound only by the length of the chain, and all other parameters don't affect the receiver in a significant way.

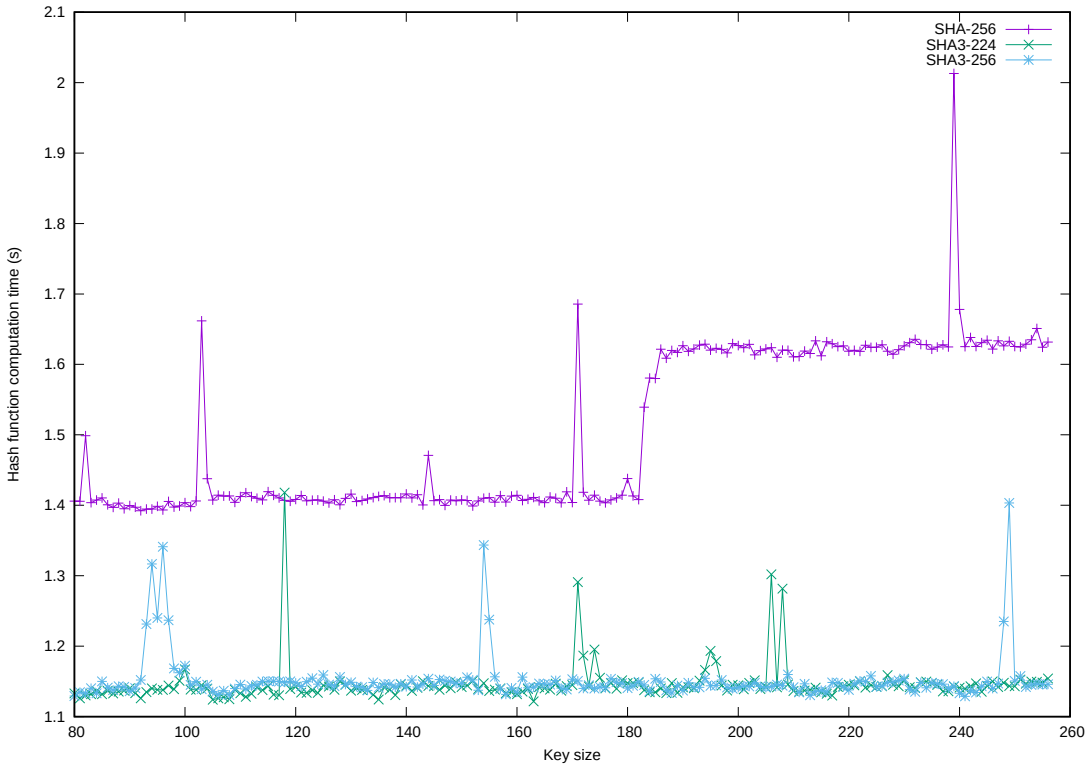

Figure 5.2: Key length vs computation time of the supported hash functions

5.3 Floating KROOTs

As part of the protocol specification described in [3], a mention is made regarding the possibility of sending DSM for KROOTs other than the one with index 0, but no further information related to how this would work are provided. In this section we dwelve deeper into the topic and make some hypothesis on how floating KROOTs could be managed effectively and efficiently.

Due to the nature of the key chain in TESLA, a key K_m sent as part of a MACK section can be authenticated against any other key K_j with index $j < m$ that has already been authenticated by the receiver. The only condition for this to work is to know how many times the receiver needs to invoke the hashing function F over the key K_m in order to obtain K_j.

Unfortunately OSNMA doesn't send explicit key indices, but being the navigation message sent at a fixed bit rate, the time of transmission of a subframe can be used to identify a key in the chain. In order to calculate the distance between the two keys, a receiver needs to consider:

- t_m: the time (in seconds) of transmission start of the subframe in which K_m is applied
- t_j: the KROOT time (in seconds) calculated as combination of KROOT WN and KROOT DOW as described below
- n_M: the number of MACK sections contained in a subframe, as sent in the NMACK field in the HKROOT section. This directly relates to the number of keys applied in a subframe as one MACK section always contains one key
- p: the relative position of key K_m in the chain starting from the first key of the first MACK section of the subframe in which K_m is sent
- NS: the number of satellites, normally set to 36 as per protocol specification

Once these parameters are known, then it's possible to calculate the distance between the keys as follows:

$$d = \frac{t_m - t_j}{30} n_M \cdot \mathrm{NS} + p \tag{5.3}$$

In plain words, this equation means that the number of keys in the chain between the root key and the key being received is composed of the sum of the number of keys used to authenticate all the previous subframes for all satellites and the number of keys that stand in between the received key and the first key in the subframe that's currently being read by the receiver. Having decomposed this result in these two parts we can now analyze how to calculate $t_m - t_j$ and p, since OSNMA doesn't send these parameters explicitly.

To calculate t_j, we can start from the week number KROOT WN and the day of week KROOT DOW associated to the KROOT that are sent in the HKROOT section. These two parameters identify the KROOT in the following way: the KROOT is the key immediately preceding the key in the chain whose application time coincides with a subframe being sent at the beginning of the day of week in the week number just mentioned.

From this, we can derive that

$$t_j = t_0 + \mathrm{WN}_j \cdot 604800 + \mathrm{DOW}_j \cdot 86400 \tag{5.4}$$

where t_0 is the GST epoch time, 604800 is the number of seconds in a week and 86400 is the number of seconds in a day.

With the same line of reasoning, we can calculate t_m by using the GST of the start of the subframe in which K_m is sent. According to [4], this is composed of two fields: the week number counted from the GST epoch, and the time of week TOW in seconds; this yields:

$$t_m = t_0 + \mathrm{WN}_m \cdot 604800 + \mathrm{TOW}_m \tag{5.5}$$

Now expanding 5.4 and 5.5 in 5.3 we obtain:

$$\begin{aligned} d &= \frac{(t_0 + \mathrm{WN}_m \cdot 604800 + \mathrm{TOW}_m) - (t_0 + \mathrm{WN}_j \cdot 604800 + \mathrm{DOW}_j \cdot 86400)}{30} n_M \cdot \mathrm{NS} + p \\ &= \frac{\mathrm{WN}_m \cdot 604800 + \mathrm{TOW}_m - \mathrm{WN}_j \cdot 604800 - \mathrm{DOW}_j \cdot 86400}{30} n_M \cdot \mathrm{NS} + p \\ &= \frac{(\mathrm{WN}_m - \mathrm{WN}_j) \cdot 604800 + (\mathrm{TOW}_m - \mathrm{DOW}_j \cdot 86400)}{30} n_M \cdot \mathrm{NS} + p \end{aligned} \tag{5.6}$$

What remains to be done now it so calculate the parameter p. In order to do

so we need to keep again into consideration the fact that the keys belonging to a single key chain are spread across satellites and that therefore consecutive keys in MACK sections sent from the same satellite are actually separated by NS keys. In addition to that, since there is no index sent within the MACK sections, the receiver needs to keep a counter l of how many keys it receives within a subframe (this counter will be reset with every new subframe). This will directly count the position of a given key within the sub-chain starting at the beginning of the subframe. This allows us to define p as

$$p = l \cdot \mathrm{NS} \tag{5.7}$$

Finally, expanding 5.7 into 5.6 we obtain:

$$\begin{aligned} d &= \frac{(\mathrm{WN}_m - \mathrm{WN}_j) \cdot 604800 + (\mathrm{TOW}_m - \mathrm{DOW}_j \cdot 86400)}{30} n_M \cdot \mathrm{NS} + l \cdot \mathrm{NS} \\ &= \left[\frac{(\mathrm{WN}_m - \mathrm{WN}_j) \cdot 604800 + (\mathrm{TOW}_m - \mathrm{DOW}_j \cdot 86400)}{30} n_M + l \right] \cdot \mathrm{NS} \end{aligned} \tag{5.8}$$

A first observation we can make regarding this result is that calculating the distance between two keys is not as straightforward as it could be for an embedded receiver. While it's true that it involves only integer arithmetic, it still requires the receiver to perform multiplications and divisions. This is true also for calculating the distance between two random keys in the chain, not just between a key in a MACK section and a root key. While the complexity of this calculation is not as high as that described above to correlate a key at the end of the chain with the root key, it is nevertheless an operation that needs to be run often, as every received key needs to be correlated to a previously authenticated key by means of its distance to it. A possible optimization could be to send an index along with the keys (both in the floating KROOT and MACK sections). This would not only reduce the receiver operations to just one single subtraction, but also make the key generation strategy transparent to the receiver (i.e. if the ground segment decides to change the way keys are spread across satellites, there would be no need to reprogram the receivers).

A second observation is that the time resolution for floating KROOT is 1 day. This means that not every key can be used as a floating KROOT, but rather only

the first key that is applied at the beginning of a Galileo day can be correctly targeted by means of the KROOT WN and KROOT DOW parameters. This means that approximately 1 out of a minimum of 103680 and 1 out of a maximum of 414720 keys can be used as floating KROOTs [*].

If we make the assumption that a new floating KROOT is broadcast and signed every day, we can then consider this result in a similar fashion as done above for the first KROOT. In particular, we can consider the number of keys sent in a day as being the maximum distance that keys will have from the nearest KROOT, and this in turn allows us to derive a loose upper bound on how long it can take under these circumstances to verify a key against a floating KROOT. Applying the same benchmark script as used in the previous section, and applying the same performance conversion devised in 5.2, we obtain the numbers in table 5.4.

Distance from KROOT	Time on Intel Core i5 (s)	Est. time on ARM (s)
103680	0.1411	34.753
414720	0.5688	140.10

Table 5.4: Upper bounds for authentication of a key against a floating KROOT

This is a definite improvement over the time it takes to authenticate a key at the end of the chain with the root key at its beginning, but depending on the receiver computational capabilities it might still be an expensive operation to perform.

5.4 Clock synchronization

5.4.1 TESLA security condition in OSNMA

One of the prerequisites of TESLA is for the clocks of receiver and sender to be loosely synchronized. More specifically, TESLA is based on a security condition that allows the receiver to identify a packet arrived safely by knowing unambiguously that the corresponding key disclosure packet has not been transmitted yet. This is because, should the opposite hold true, an attacker could have received the

[*] these two numbers are the number of keys sent in a day in the case where either 1 or 4 keys are consumed in a subframe, calculated by multiplying the number of keys in a subframe by the number of satellites and by the number of subframes in a day: $\frac{86400}{30} 36 \cdot \text{NMACK}$

key disclosure packet and forged a message using that key; without the guarantee specified in the security condition the receiver has no way to detect this kind of attack.

In OSNMA it's still necessary to meet this condition in order to guarantee the security of the whole algorithm. We'll describe next a possible attack that relies on the clock of the receiver being undefinitely out of sync with system time.

5.4.2 An attack against clock synchronization

The scenario is composed of the following actors:

- a legitimate Galileo satellite, sending authenticated navigation data following the OSNMA specification. The clock of this sender is perfectly synchronized with the central system time
- a Galileo receiver with an unknown clock drift δ_t upper bounded by a known fixed value δ_{max}
- an attacker capable of receiving Galileo navigation data and transmit a custom signal resembling that of one or more Galileo satellites, and containing ad-hoc forged data.

In this scenario the attacker is free to modify the received data, but is limited in the following:

- it cannot forge the root key transmitted in the HKROOT section. This is based on the security of the asymmetric scheme used to authenticate the root key
- for the same reason, it cannot forge the transmission time of mentioned key. The transmission time (in form of the Galileo week number and day of week) is included in the signature and such forgery would be detected with the same high probability as a forged key
- if it wishes to reuse a key of the chain, it must send it in the right order with respect to the others and with the right time. This is because the transmission time of a key univocally determines its index in the chain, and the transmission time is also authenticated in the MACs sent in the MACK sections. Should the key be sent out of order (i.e. with the wrong time), the receiver wouldn't be able to authenticate it against the root key

Another assumption we make is that the receiver is already sending a forged navigation message before the receiver starts to receive on the same frequency the attacker is transmitting. The message sent by the attacker contains a forged set of ephemeris crafted so that the receiver would derive a set of coordinates different than its real ones. This data is authenticated using the OSNMA protocol as follows:

- the attacker receives the legitimate signal at time t_k, reading the keys contained in the MACK sections
- the attacker uses the keys it read in the same orders as they were received to authenticate the forged data
- the forged data is transmitted at time $t_k + \delta_f$ with a power sufficient to spoof the legitimate signal

Here δ_f is a delay that accounts for the fact that a key is necessarily read after the corresponding MACs, but the attacker needs to replace the MACs *after* having read the key, so the resulting message needs to be delayed by a fixed amount of time that allows the attacker to perform the attack and build a valid navigation message.

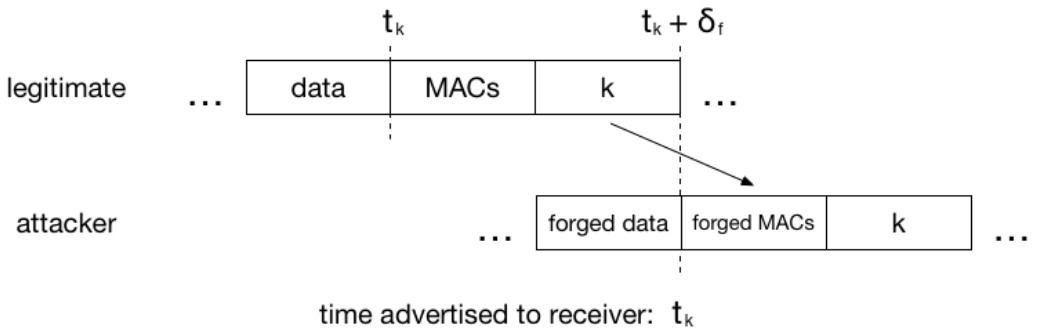

Figure 5.3: Possible forging attack that relies on lack of clock synchronization

Under these conditions we can see what happens on the receiver side:

- When it wakes up, the receiver immediately starts to acquire and track the forged signal. Since its clock offers no guarantees of synchronization against system time, the receiver is not capable of spotting that the received signal is delayed.

- At first the receiver will decode and authenticate the root key; this operation will succeed since the attacker is just sending a delayed version of the original data.

- Once the root key is authenticated, the receiver starts to receive the navigation message and the respective MACK sections. The receiver at some point will verify that the GST advertised in the subframe matches the time that has been signed by the keys it's receiveing; this operation will also succeed since at this point the receiver has no knowledge of the true system time and the attacker is correctly building the signatures following the protocol.

- The receiver will proceed by authenticating the key it received against the root key. This step will also be successful as no forgery is necessary on the time parameters of the navigation message, and the order of the keys is being preserved by the attacker.

As shown by the above example, without guarantee of synchronization between the receiver and sender's clocks, and without an upper bound on the drift between the two, a spoofing attack can go undetected even in the presence of data authentication in case a receiver starts receiving without prior knowledge of any system parameter. We'll describe next how this risk could be mitigated.

5.4.3 Preventing forgery attacks

The problem just described is addressed in [2] as a prerequisite to the algorithm: i.e. TESLA doesn't provide any mechanism to synchronize the clocks beforehand, but rather demands that an upper bound on the drift is known. The problem with real hardware is that Galileo needs to support a wide range of receivers, some of which sport clocks with poor performances. Moreover, no assumption can be made on the operating mode of the receiver: for example, a receiver could stay offline and accumulate drift for indefinite amounts of time. From this standpoint, we identify three approaches that can help make sure the security conditions are met, all of which we'll describe next: either we support Galileo operations with external clock sources, we require hardware with a certain level of guaranteed performance, or we provide guidelines for receivers to maintain the clock drift under a specific threshold.

An example of the first case is to use a network protocol such as NTP to perform an initial time synchronization before starting to receive and authenticate Galileo data. This is an approach that can suit some receivers, but for other use cases it's not viable. If we consider for example a receiver mounted on a boat to provide assistance to deep sea navigation, requiring Internet connectivity in such an environment might be too hard a constraint.

In the second case, we could require receivers that wish to authenticate data to have on-board clocks with a precision similar to that of the clocks mounted on the space vehicles; this way we're guaranteed by the performance match that the drift will stay within constant bounds. While development of high precision, integrated atomic clocks in under way, their cost is still prohibitive for the mass market.

The third approach instead puts control in the hand of the receiver and could therefore suit a broader set of receivers than the other two approaches. The main idea is that ultimately what's important is to keep a receiver's clock drift under control, and this can be achieved by using Galileo itself (or any other GNSS for that matter) provided certain receiver parameters are known. We can illustrate the idea with a general framework, and then analyze in more detail some specific configurations that might suit the case of OSNMA.

Let's assume a receiver using a clock with a known relative drift of d. Let's also assume the receiver starts its life with the clock perfectly synchronized with GST. We know that TESLA requires sender and receiver not to drift apart more than an upper bound δ_{max}. We can therefore derive after how much time the threshold will be reached with:

$$t_{th} = \frac{\delta_{max}\,\mathrm{s}}{86400 \cdot d\,\mathrm{ppm}} \tag{5.9}$$

This number gives a lower bound on how often the receiver needs to synchronize itself against GST. If the receiver adjusts its time *at least* every t_{th} s, then the security condition can be met and the receiver is assured that data can either be authenticated correctly, or a spoofing attack can be identified.

Starting from this result, we can then derive a value of the disclosure lag δ_{max} that would guarantee security in OSNMA. As a reminder, this value is chosen

so that receivers have a guarantee that whenever they receive a MAC the corresponding key hasn't been disclosed yet. This definition doesn't define precisely the time boundaries of such a lag. The beginning of the disclosure lag can be computed starting from the beginning of transmission of either the whole MACK section, or just the key. The end of the disclosure lag can be marked unambiguously at the time when the transmission of the MACK section ends. To resolve the ambiguity, we will calculate the lag for the shortest amount of time (i.e. the time it takes to transmit a key rather than the time it takes to transmit the whole section); this represents the worst case as shorter disclosure lags mean more stringent synchronicity requirements, so we can consider the other case a relaxation of this analysis.

According to this definition, the disclosure lag is equivalent to the time it takes to transmit a key. Additionally, in OSNMA the disclosure lag can take different values for a fixed key size: in standard conditions, the key is sent just after the corresponding set of MACs it generated, so the disclosure lag is just the time it takes to transmit a key. The specification also mentions the possibility of using SLMAC (shorthand for "slow MAC"), in which the key is disclosed at some subframes of distance from the MAC. We'll analyze this case later as it builds upon the standard case.

According to the OSNMA specifications, the length of a key ranges between 80 and 256 bits. The I/NAV navigation message to which OSNMA applies is transmitted at a nominal rate of 120bps, but the MACK section occupies only 32bit of a nominal 2s page, making its effective transmission rate equal to 16bps. Therefore the time it takes to transmit a whole key is between 5s and 16s.

From this result we can try to understand how fast a clock can achieve the maximum drift depending on its accuracy. Since there's no standard we can take some sample values to see how they affect the security condition. A normal value for clock accuracy for embedded devices is in the order of 10ppm; we consider along that two higher precision marks to understand how higher quality clocks could improve the receiver operation.

Clock precision [ppm]	Time to max drift [d] (80bit key)	Time to max drift [d] (256bit key)
10	5.787	18.52

Clock precision [ppm]	Time to max drift [d] (80bit key)	Time to max drift [d] (256bit key)
1	57.87	185.2
0.01	578.7	1852

Table 5.5: Days to reach maximum thresholds with different clock drift rates

We can see that, for example, a clock with a precision in the order of 10ppm would reach the maximum allowed clock drift in less than a week in case of keys that are 80bit long, and in a little less than 3 weeks in case of keys that are 256bit long. This can also be seen as the maximum amount of time a receiver can stay offline before the clock drift prevents it to safely authenticate the data it receives. The solution for a receiver is then to be designed around this security requirement so that it keeps track of the time elapsed since last clock synchronization and, depending on the nominal precision of its clock, makes sure that it checks its drift from system time with a period shorter than the upper bound just mentioned.

As stated before, the OSNMA specification describes the possibility of having some keys sent with some delay with respect to the MAC they generated. This concept is named SLMAC in the specification, and it's supported on a single set of data with two possible delays: one or ten subframes. This amount to an additional delay of 30s to 300s. Theoretically, these disclosure lags would allow for the delays reported in Table 5.6.

	Clock precision [ppm]		
	10	1	0.01
80bit key, 30s delay	40.50	405.0	4050
256bit key, 30s delay	53.24	532.4	5324
80bit key, 300s delay	353.0	3530	3.530×10^4
256bit key, 300s delay	365.7	3657	3.657×10^4

Table 5.6: Days to reach maximum thresholds to guarantee security of SLMACs

As we can see, with a disclosure lag of five minutes we can allow receivers with relatively low accuracy clocks to remain out of sync for around a year no matter

the size of the key.

6

OSNMA: Receiver Operations

In this chapter we'll describe a possible state machine for a receiver that supports OSNMA. We make the assumption that the receiver CPU is single core and single threaded. That is, we'll describe a purely sequential and synchronous state machine that doesn't involve any form of concurrency. The input of this state machine is the decoded navigation message. In other words, we are not concerned here about the operations of the receiver's radio frontend, but just of those of the data processor, and specifically we're interested in understanding how the authentication operations affect the complexity of a receiver's firmware.

A first discrimination a receiver should do is if in its current state the amount of information it holds is sufficient to guarantee it only receives legitimate data. This amounts to having an authenticated key of the current chain and knowing that the last period of inactivity doesn't go beyond the threshold calculated as in the previous chapter. In the first case the receiver can proceed to receive and authenticate the navigation message, and to solve the PVT equations with the authenticated data. If not, the receiver must enter a bootstrap phase where the conditions described above are met.

6.1 Authentication bootstrap

As part of this phase, the receiver needs to bring itself to meet two conditions, or fail if that's not possible: a root key is present, and the clock is guaranteed to not have drifted too much since the last synchronization. These two conditions are independent, but we'll see later that in order to synchronize the clocks an authenticated key must be in possession of the receiver already.

A procedure to bring the receiver in a state where authenticated navigation is possible is as follows:

- first the receiver should check the presence of an authenticated key in its permanent storage. Should the key be absent, the receiver can proceed with the next step; otherwise, it can proceed with verifying the clock drift
- verify that the chain in force is not in the EOC (End Of Chain) state, in which case the receiver should wait to receive the root key for the next chain in force as it cannot distinguish if Galileo is sending the root key for the ending chain or the root key for the new chain
- verify that the chain is not being revoked, in which case the receiver should wait until the new root key is sent, identified by a combination of *NMA status* and *Chain status* of "Operational" and "CREV".
- read the HKROOT: here the receiver builds up all the information contained in the HKROOT section. This is important for two reasons: get the root key that will allow to bootstrap the auhtentication process, and read the system parameters that will be used throughout the whole session
- authenticate the KROOT: this is done by using the public key identified by the public key ID in the headers section to hash a message containing the KROOT, as described in [3], section 5.1
  - if the verification succeeds, then the authenticated key is marked as such and stored in permanent memory together with the other information contained in the header, and the receiver can proceed to the next step
  - if the verification fails, then depending on how stringent the security requirements are, the receiver could either proceed to solving the navigation problem without authentication, or abort its operation. In either case a notification should be sent to the end user

- at this point clock drift should be checked. Based on nominal clock precision, key size as defined in the KS field of the NMA header, and Equation 5.9 the receiver should estimate for how long it could stay safely offline.

  - if the difference between current time and the last time it connected to Galileo (which should have been stored) is below the threshold for normal operations (i.e. keys sent just after the corresponding MAC), the receiver is safe to step into receiving and authenticating navigation data.
  - if not, a procedure of clock synchronization using SLMACs should be initiated, which we'll describe next. Once this procedure is completed and clock drift has been restored to a level where standard authentication operations are safe, then the receiver should step into the navigation state.

6.2 SLMAC clock synchronization

If the offline period has been too long to guarantee that the security condition is met for keys that are sent just after the corresponding MAC, the receiver can try to reduce the drift by reading the GST information from a subframe that is being authenticated using a SLMAC. This can be done after the receiver has verified if the security condition for SLMAC holds:

- the receiver should first verify if its offline period allows to trust keys delayed by 30s
- if this is not the case, then the receiver should check if the offline period allows to trust keys delayed by 300s
- if none of these conditions are met then OSNMA cannot be safely used. Depending on how the receiver is configured it should either abort its operations or continue by solving the navigation problem without authenticating the data. In either case a notification should be shown to the end user

Once the receiver has determined which key delay meets the security constraint, it should initiate the clock synchronization procedure as follows:

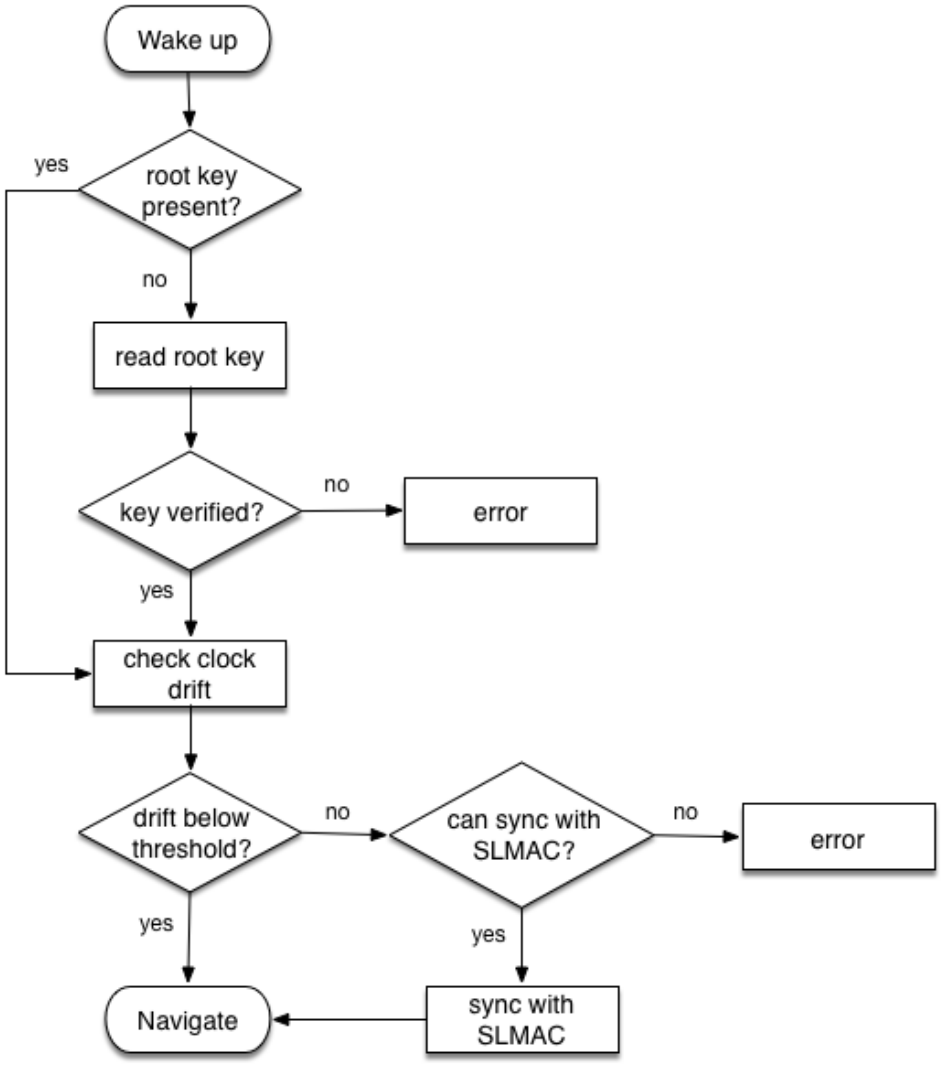

Figure 6.1: Decision tree for OSNMA authentication bootstrap

- first, it should keep reading subframes until a MAC with an ADKD value of 11 or 12 is transmitted (depending on the key delayed identified: ADKD=11 for 30s delay, ADKD=12 for 300s)

- while reading subframes, the receiver should keep track of the GST values read in WT5 (GST WN and GST TOW as described in [4]). Together with this, the receiver should mark the time at which the GST was received (i.e. the local time of beginning of the subframe reception). This is important

as the receiver must be able to account for the time that pass between advertisement of GST and the actual clock synchronization

- once the MAC with ADKD set to 11 or 12 has been read, the receiver must wait either 1 or 10 subframes, and then read the key being transmitted at that time

- once the key has been received, it can be authenticated against the latest authenticated key stored in permanent memory (it being the root key or any other more recent key that has been verified against the root key). This is done by applying the selected hash function a number of times as described in the authentication bootstrap paragraph above

- if the key is successfully verified, the receiver can proceed to synchronize its clock with the GST it read *while reading the authenticated key*. Authentication of the GST is guaranteed by the fact that the advertised GST is directly used to compute the key index, so a forged timestamp would result in the key not being verified. At the same time, the receiver should overwrite the last authenticated key in permanent storage with the one just verified, to help speeding up authentication of subsequent keys. Once the clock is synchronized, the receiver can step into the navigation state

- if the key cannot be verified, depending on its settings, the receiver should either abort its operations or proceed to the navigation state with disabled authentication. In either case the end user should be notified.

6.3 Authenticated navigation phase

If the receiver has made sure of complying with the security condition of TESLA it can enter the navigation phase. The goal of this phase is to solve the PVT equation using authenticated data received from satellites. For this to be accomplished the navigation message needs to be received from a number of in-view satellites, together with their authentication codes and the keys that generated them. Given the nature of how data is transmitted, two aspects need to be considered. First, an efficient way to store and access the data just mentioned that accounts for piece-wise reception of information. Second, a state machine to process the received data and handle error cases.

To understand the problem of storing the data related to the navigation message and its authentication, it's useful to analyze the lifecycle of these two components. Starting from how OSNMA groups data to be authenticated with a single MAC, we distinguish three main categories: ephemeris data, almanac data and Galileo-GPS conversion parameters. In a 30s cycle, the navigation message includes all the ephemeris for the transmitting satellite, the conversion parameters, and almanac data for one and a half satellites out of 36. In the same time frame, depending on the configuration, 1 to 15 MACs can be transmitted. This means that in some cases the receiver could receive data in a subframe and its authentication in a subsequent subframe, or the other way around. In any case this means that the receiver needs to have a local cache that captures the following:

- authentication parameters contained in the HKROOT header section
- individual navigation message fields
- authentication codes, their associated time and position within a subframe
- keys associated to the authentication codes
- status of the validity of a key
- status of the verification of an authentication code

The navigation message fields amount at least to:

- ephemeris for 36 satellites
- almanac for 36 satellites
- remaining bits of a subframe that are not captured by other fields

In case the receiver is capable of receiving GPS signals and wishes to authenticate them, then the storage requirement includes also the authentication codes relative to the GPS navigation message, for all the possible 35 GPS satellites Galileo can authenticate.

One important aspect to consider that might affect the efficiency of the storage mechanism is the fact that one authentication code applies in general to more than one field. To avoid duplicating the authentication codes for each field, a pointers-based structure might be implemented, in which for each field just a

pointer to the data structure storing the actual MAC and contextual timing and position information is stored.

Another important tradeoff that must be considered when designing the storage of the receiver is related to the fact that Galileo can authenticate also a whole full subframe; here the two opposing forces are memory and complexity. On one side, one could allow for a complete copy of the subframe to be stored independently from the other fields. The second approach is to collect the bits of the subframe that are not included in the other stored fields, and have the CPU recompute the subframe structure prior to authentication. This last approach has the advantage of saving memory, but requires some computation on the processor side before the subframe could be authenticated. The first approach instead removes the need of CPU cycles at the expense of more memory used and some duplicated data. On the other hand, this approach also allows the receiver to store ephemeris and almanac data independently from the data received in the last subframe (e.g. the receiver might keep a verified version of the individual fields for use in the PVT solution, and at the same time authenticate the last subframe with the option of discarding it in case authentication fails without losing already verified data).

Once the data structure is defined, it's useful to discuss how the receiver should fill the actual data in. In general it's important to realize that some of the data presented above needs to be computed by the receiver, which needs to make sure enough CPU cylces are scheduled at the right time, without running into the risk of blocking other operations. From this point of view, one approach is to perform data processing operations only at the boundaries of logical parts of the navigation message, namingly at the end of a page, at the end of a subframe or at the end of a frame, as suggested in [10]. Processing the navigation data at the end of a 2s page has the potential advantage of making data to be authenticated available faster, at the expense of the need for more complex logic for handling the different types of information included in different pages. This involves having different parsers for each word type, code for routing the incoming data bits to the relevant parser, and some state management code that keeps track of the kind of word being sent at any specific moment.

Processing the data at the boundary of a subframe, on the other hand, can allow for a simpler receiver implementation as the data received in a complete

subframe has always the same structure, and only the satellite's index in the almanac data is different. The downside of this is that data would be available only in 30s intervals.

Moreover, parsing received MACK sections might be more easily done at the subframe boundaries, since that's the only boundary over which a receiver can be sure of the structure of the data received; in other words, parsing data at the page level would require the receiver to understand if it received one or more whole MACs and associated keys of it's in the middle of receiving one of those entities.

Assuming then that the data processing and authentication happens at the boundary of a subframe, we can proceed with describing a possible algorithm that ensures the receiver solves the PVT problem using only authenticated data:

- the first step is to parse the data and fill out the fields in the memory model described above. Data shouldn't live yet in permanent storage as it hasn't been authenticated yet
- as a second step the receiver should decide if the chain currently in force is reaching its end or if it's being revoked. In both cases, the receiver should start to receive the new DSM-KROOT and authenticate the new root key before resuming nominal operations.
- in case the chain in force is in nominal mode, the receiver can scan the list of keys it received, and for each key it proceeds with the following:
  - authenticate the key against the last available authenticated key. This involves performing a number of hash function calls according to the algorithm and constraints described earlier
  - if a key is not valid, the receiver should discard all the MACs associated to that key as the receiver cannot guarantee their authenticity
- after the keys have been verified, the receiver can proceed to scan the MACs, and for each available MAC associated to a valid key perform the following:
  - looking at the ADKD associated to the MAC, the receiver should try to put together all the fields associated to the satellite authenticated by the MAC

  - if not all required data has been received yet, the data authentication step should be skipped, and the next MAC should be analyzed
  - if all fields have instead been received, the receiver should build the tag according to the OSNMA specification and authenticate it using the parameters specified by the protocol (in particular, hash function and hash truncation size)
  - if the above authentication step succeeds, the authenticated fields should be stored in permanent storage and marked as authenticated; moreover, the used MAC can be safely deleted
  - if the authentication fails, the data should be deleted as it cannot be securely used by the receiver. Depending on the design of the receiver, at this point a notification should be sent to the user

- after all available MACs have been processed, the receiver might still be in possession of unauthenticated data or unused MACs due to missing data. The data should be kept in a queue for use during the next processing cycle
- if the number of authenticated fields in permanent memory is sufficient to solve the PVT equations, the receiver should move forward with PVT determination

6.3.1 Managing exceptions

One important aspect to be considered in order to avoid a certain class of attacks against robustness of the receiver is how to handle exceptions in case the received data represents an inconsistent state for the receiver. For example, an attacker might try to crash the receiver by advertising keys of a size and a number of MACK sections in a subframe that represent an impossibility given the constraint of the general size of the MACK section for a subframe. In this case the receiver should be able to spot the inconsistency and make sure it doesn't run into a class of problems like segmentation faults or buffer over- and underflow.

In specific, the following constraints should hold at any time during the receiver operations:

- no restricted value for any of these fields should have been received:

  - *NMA status*
  - *Chain status*
  - *Number of blocks (NB)* of the DSM
  - *Number of MACKs (NMACK)*
  - *Hash function (HF)*
  - *MAC function (MF)*
  - *Key size (KS)*
  - *MAC size (MS)*
  - *ADKD* in the MACK sections

- depending on the ADKD value received, the corresponding IOD field could assume fixed values. The receiver should ignore the IOD field in those cases.
- the combination of key size *KS*, MAC size *MS* and number of MACK sections per subframe *NM* should at any time satisfy the equation:

$$\left\lfloor \frac{480 - \mathrm{KS}}{\mathrm{MS} + 16} \right\rfloor \geq 1$$

  where 480 is the size of the MACK section, 16 the size of the MAC-Info section within it and $\lfloor x \rfloor$ is the floor operation (the largest integer no bigger than x)
- the *Number of blocks* parameter shouldn't be set to 0
- the *Block ID* advertised in the DSM Header should not be larger than the advertised *Number of blocks*
- the combination of *Chain status* and *NMA status* fields should be within the following set:
  - Nominal - Operational
  - EOC - Operational
  - CREV - DU
  - CREV - Operational

Whenever one of these conditions is not respected, the receiver should, depending on the design, either abort its operations or proceed without authentication enabled and inform the user about the incident.

7 Conclusion

7.1 Future work

During the analysis a few aspects were highlighted that could benefit from future research. The first of them is related to the fact that even in an ideal situation where no errors are introduced in the transmission of data, authentication of the navigation message takes an interval of time in the order of minutes. This is a constraint that might be too stringent for certain applications, and it would be interesting to understand what possibilities the current message structure offers in order to improve the timinthe timing of a first authenticated position fix.

Another topic that requires some discussion is related to the error conditions in the protocol. Generally from the analysis a tradeoff emerged: either security is mandatory or it's considered as a mere "upgrade" to the unauthenticated service. Depending on the perspective taken, treatment of error conditions can follow very different paths. In the first case, with an extreme approach the receiver interprets every kind of error as an attempt to spoof the data, which requires as a countermeasure a complete discard of the information received from a particular satellite. In the second case, the receiver still identifies the security breach but realizes that a complete shutdown of the navigation operation is not possible. In that case, the receiver could either downgrade communication to the default non-

secure protocol, or it could put the choice in the hands of the user (one example of this approach are browsers that fail to authenticate a server exposed through an HTTPS endpoint, in which case the user is asked if they want to proceed or not). Both of these choices offer advantages and drawbacks. A complete discussion of those could provide clearer guidelines to receiver implementors, and eventually identify more sophisticated approaches to identify real security breaches and recoverable scenarios.

To facilitate the analysis of these and other aspects, a software simulator could be of great benefit. As of the time of writing, several simulators exist for GPS and to a lesser extent for Galileo - both on the receiver and on the transmitter side. Nevertheless, none of them yet implements OSNMA. A project that implements it would be of aid in performing more in-depth analysis of the attack patterns and the potential implementation problems that hardware implementors might face.

Finally, one overarching aspect emerged in most of the point touched by the analysis conducted in this document. It is clear that the security of the overall protocol, as it is today, relies heavily on implementors to follow the specification guidelines in order for it to be effective against the attacks it promises to block. As an example of this, let's consider clock synchronization: should receivers ignore or miscalculate the drift threshold, the receiver would be exposed to spoofing attacks that might be even subtler to identify as the authentication layer might give a false sense of security. This aspect is even more relevant considering that, for example, in the case of clock synchronization the receiver requires certain external conditions to be met (for example, that the device doesn't remain inactive for too long a period). While in practice this might not be a problem in a high percentage of cases, it is still an architectural constraint that might consitute a roadblock for certain applications. From this point of view, one way to prevent this to happen is to perform research on ways to overcome the external constraint by either using external aid in bringing the receiver in a state that matches the security conditions, or by working on an expansion of the protocol that could remove the constraint altogether. One idea could be for example to provide some form of authenticated time the receiver can use as a starting point to reduce the drift in

clock synchronization. The time could be provided in a fashion similar to that of the DSM-KROOT, at a slower pace but authenticated in a way that's infeasible to spoof.

7.2 Final words

While analyzing the OSNMA protocol for Galileo, a few distinct points became clear. This implementation of the TESLA protocol had a hard challenge in front: adding a layer of security on top of a protocol that is unidirectional with no possibility of making it bi-directional and with an already low transmission rate. All of this required an effort that resulted in some adaptations of the original security scheme in order to support aspects like the presence of multiple transmitters, which weren't part of the design requirements of TESLA.

All of this carries along more complexity on the side of the receiver, which needs to be able to handle a more elaborated state machine and handle more possible error states than before. Another trend that emerges from the analysis of the computational requirements for the protocol is in general the availability of more computational resources to perform a high number of cryptographic operations in a short period of time. At the same time, this requirement comes along with the need of not increasing the amount of power consumed in performing authentication of the navigation message. In other words, the need for security and flexibility of OSNMA needs to be traded off with the need of keeping the energy profile of the receiver compelling for portable use. At the same time, it has been clearly exposed how higher quality local clocks might also be required if security of radio navigation is to become a commodity; the lower the quality, the higher the chances that an individual receiver might fall out of the acceptable range of clock drift.

This panorama suggests two opposite points of views, which at the same time don't entirely contradict each other. On one side, one could try to minimize the impact of this security implementation on the current design of hardware embedded receivers. This is doable by being clear on the minimum security requirements (for example, in terms of clock synchronization), and by having some support on the transmitter side on the strategy for the transmission of

secure information (for example, the amount and frequency of transmitted floating KROOTs).

On the other side, one could take the introduction of such protocols as a sign of times to come, which speak of the need for increased receiver complexity and computational power. From this point of view, the power is in the hands of the implementor to find innovative ways to build receivers capable of complying to more demanding applications while retaining their appeal to the market.

References

[1] *European GNSS Service Centre: Constellation Information*, accessed 12 Feb 2018. [Online]. Available: https://www.gsc-europa.eu/system-status/Constellation-Information

[2] A. Perrig, R. Canetti, J. D. Tygar, and D. Song, "Efficient authentication and signing of multicast streams over lossy channels," in *Proceeding 2000 IEEE Symposium on Security and Privacy. S P 2000*, 2000, pp. 56–73.

[3] T. A. P. W. G. S. J. S. C. S. D. B. O. P. I. Fernandez, V. Rijmen, "Galileo navigation message authentication specification for signal-in-space testing - v1.0," November 2016, part of the Tender specifications (Annex I) published in the GNSS opportunity database and accessible at https://bit.ly/2HhFPsE - published 19 May 2017, accessed 08 Apr 2018.

[4] *Galileo OS - SIS ICD v1.3*, December 2016, published by the European Global Navigation Satellite Systems Agency, available through the European GNSS Service Centre website: https://bit.ly/2uUVFqu. [Online]. Available: https://bit.ly/2HghyTH

[5] *SiSoftware Official Live Ranker: Details for Component Intel Core i5-7300U*, published 23 Feb 2018, accessed 23 Feb 2018. [Online]. Available: https://bit.ly/2HX72jy

[6] *STA8088EXG: Flexible GPS/Galileo/Glonass/QZSS Receiver with powerful processing (ARM9)*. [Online]. Available: https://bit.ly/2JtysPx

[7] *Mediatek MT2523D/MT2523G product overview*, accessed 20 Feb 2018. [Online]. Available: https://bit.ly/2pzIa9M

[8] *Dhrystone and MIPs performance of ARM processors*, published 28 July 2010, accessed 08 Apr 2018. [Online]. Available: https://bit.ly/2IBzAPH

[9] A. L. Richard York, "Benchmarking in context: Dhrystone," March 2002, accessed 23 Feb 2018. [Online]. Available: https://bit.ly/2DNZbSG

[10] J. Nurmi, E. Lohan, S. Sand, and H. Hurskainen, *GALILEO Positioning Technology*, ser. Signals and Communication Technology. Springer Netherlands, 2014. [Online]. Available: https://books.google.de/books?id=ktOEBAAAQBAJ

Acknowledgments

I wish to thank a few amazing individuals that guided and supported me through the challenge of writing a master thesis in the midst of a job change, a newborn relationship, and a move. The first is Gianluca Caparra, who showed admirable patience and dedication in holding my back. Gianluca, I'm grateful for life for you always finding time to answer my questions, jump into skype calls, and review my early drafts. I truly couldn't have made it without you.

Second, I would like to thank Nicola Laurenti for having given me the freedom of doing my research from abroad, without a trace of micromanagement and without questioning my steps (even after dropping the ball for two years and showing up for the decisive retry). Nicola, I'm extremely grateful for you having made the process easier and more enjoyable.

I also want to thank my parents, who also never questioned my choices but rather gave me the credit and trust I needed even if it was hard to see where I was going. At the same time, I'm grateful to them because they never gave up but kept the thought of this degree in my head for so many years. Giorgio and Lorena, I owe my resilience (even if sometimes it borders with stubborness) to you.

In addition, my gratefulness to those friends, near and far, that inspired and moved me over these years. In particular, Nikos and Mari: I'm grateful and honored to have you as friends, and grateful for the challenges and tough conversations we went through together.

Finally, but most importantly, I want to thank the star that since some time is brightening my path. Sarah, you're the most wonderful partner anyone could ever desire. If I was to mention all the things I'm grateful for, the pages of this work won't suffice to contain them: for one, I truly couldn't have gone so far so fast without your unconditional support, and I'm grateful for you having walked the hard path with me. Even as I write these words I can't help feeling moved at the miracle of us. Thank you for being as you are, you made true love a reality

in my life.