Independent Time Synchronization for Resilient GNSS Receivers

Ignacio Fernandez-Hernandez†, Todd Walter*, Andrew Neish*, C. O'Driscoll§

*Stanford University, †European Commission, § Independent Consultant

Abstract

This paper analyses time synchronization schemes for GNSS receivers that require time-delayed authentication. We review the synchronization process as defined for the TESLA (Timed-Efficient Stream Loss-Tolerant Authentication) protocol current literature and propose how it can be adapted for one-way communication systems like GNSS. The receiver to be synchronized at a given synchronization event *S* needs not only an independent reference time t_{ref}^{S}, but also a bound *B* of the reference time uncertainty. With these two parameters, the loose time synchronization requirement of the protocol T_L and the signal-in-space time t_{sig}^{S}, the receiver can start up the protocol. We analyse different startup cases and derive preconditions based on t_{ref}^{S}, t_{sig}^{S} ,T_L, and *B*, and propose secure startup procedures that prevent data forging and other anomalous situations. We then analyse the practical aspects of GNSS receiver synchronization for network applications and aviation, and finalise with some conclusions. The conclusions can be extrapolated to other data or signal authentication protocols based on the delayed disclosure of cryptographic information.

Introduction

Timing is one of the key services provided by GNSS. The time-of-arrival scheme at the heart of GPS and also used in all other core GNSS constellations, allows receivers to obtain time to nanosecond-level accuracy without any prior time synchronization. However, having an independent time reference, even if less accurate than GNSS, can make receivers more resilient against current and future spoofing threats. Many signal and data authentication schemes proposed or under development by GNSS are based on a delayed disclosure of cryptographic information. This approach is vulnerable to spoofed GNSS signals that are able to fool the user clock into believing that the current time is sufficiently behind the actual time. Therefore, a separate means of verifying time synchronization that is independent of GNSS is required. This paper examines the use of receiver independent time synchronization in order to support authentication schemes.

The delayed disclosure scheme for authentication is currently proposed by GPS´s CHIMERA [1] based on signal watermarking [2], and SBAS data authentication [3], and required for Galileo's OSNMA [4]. Not only will independent time synchronization of GNSS receivers be required for the future GNSS authentication features, but it also can increase overall receiver resilience. However, secure independent time synchronization requirements and processes for GNSS receivers have not been thoroughly analysed in GNSS literature. This paper looks at both theoretical and practical aspects of receiver independent time synchronization.

From the theoretical point of view, the TESLA (Timed-Efficient Stream Loss-Tolerant Authentication) protocol [5] is studied as a baseline. TESLA has been adapted for the current Galileo OSNMA scheme [4] and is under study for SBAS authentication [3] due to its suitability for a low-bandwidth channel such as GNSS. It requires the sender and the receiver be loosely synchronized in order to guarantee data authenticity. The receiver must verify that the received data is not delayed by more than a given amount of time defined by the TESLA loose time synchronization requirement T_L. The protocol originally proposed client-server synchronization schemes based on a two-way communication channel being available in network implementations. However, for GNSS, the receiver needs an external means of synchronization at startup, or a sufficiently accurate internal clock reference independent from GNSS. Even if we use authentication schemes not based on time-delayed asymmetry, such as digital signatures, loose time synchronization may be required if data replay attacks are to be protected. For example, in order to ensure that expired GNSS ephemeris data are not replayed and used, the receiver would need a time reference on the order of hours. A tighter requirement would apply in the case of corrections with a shorter validity time, such as SBAS corrections, or GNSS satellite flags.

The paper presents first an overview of the TESLA protocol. This overview is followed by a discussion on how to adapt TESLA, as described in previous non-GNSS literature, to GNSS. We analyse different startup cases and derive preconditions based on the independent reference time, the GNSS signal time, the loose time synchronization requirement T_L, and the reference time error bound B, analysing false alerts and missed detections. With these preconditions, we propose secure startup procedures that prevent data forging and protect against other possible anomalous situations.

On the practical side, the paper analyses how real receivers can deal with independent time synchronization. This includes network-assisted receivers and standalone receivers. It characterizes the initial time uncertainty based on the external time sources, such as mobile networks and other systems, and the drifts of internal real-time clocks from available devices in the market, and compare them with the T_L requied by current authentication protocols. The standalone receiver analysis will be primarily aimed at avionics receivers, for which a startup synchronization process and reference time requirements will be proposed. This proposal can support future avionics standards including data authentication protocols.

TESLA Overview

The TESLA protocol is a perfect example of a delayed disclosure scheme. The user is initially provided with data and a Message Authentication Code (MAC) which is a function of the data and a as yet unknown key. The server that provided the data and MAC later releases the key and the user can verify that the previously received MAC matches the functional output by combining the data and the key. It is the delay between the release of the first two elements (the data and the MAC) and the final element (the key) that provides the authentication (along with a separate technique to validate the key). At the time the data and MAC are released, only the server knows the key. Once the key has been released, anyone can create a matching data and MAC combination with that key. But, up until the key was released, only the server could make such a combination. The receiver therefore needs to know over what period the data and MAC combination can be safely received for later authentication, and when the key has been made public. One should not trust the data-MAC combination broadcast after that time. If the receiver and server are synchronized, then the user can confidently authenticate the data. However, if the receiver's clock is too slow, then they may be fooled

into accepting a data and MAC combination that is received after the key has been made public. Therefore, it is imperative that the user's clock not be allowed to fall too far behind the server's clock.

Differences in Synchronization between Standard TESLA and GNSS-TESLA

For the TESLA protocol we will base our analysis on two references, [5] and [6]. According to [5], the loose time synchronization requirement implies that the receiver time t_r and the server time t_s have to be loosely synchronized within a boundary D, such that

$$t_s - t_r < \Delta \tag{1}$$

Notice that (1) does not account for the absolute value of the error, which would result in $| t_s - t_r | < \Delta$, as only the server is broadcasting information. Therefore, the lack of synchronization only leads to a data forging attack when the receiver time t_r is *too far behind* the server time t_s, as this would allow an adversary to receive the time-delayed key in advance from the server and forge the message. In the TESLA protocol as defined in [5], the receiver and server can re-synchronize if necessary to maintain the loose time synchronization within the time boundaries required. It proposes a synchronization protocol whereby the receiver sends a time request with a salt, to the server, and the server sends back its time tag and the salt digitally signed, ensuring secure time transfer. The receiver can then check that the difference is below the loose time requirement for the TESLA protocol to function. The protocol assumes no clock drift, but proposes to resynchronize through the mentioned procedure regularly.

Alternatively, for the TESLA protocol defined in [6] the loose time synchronization requirement is defined by the "safe packet test", which consists of the following steps:

- The receiver receives a packet P_j and notes the local time T, and determines an upper bound on the true time as: $t_{true} \leq t_j = T + D(T)$, where $D(T)$ is a known time varying bound on the reference time.
- The receiver then computes the maximum possible key index $i_{\max}$ that could be known given that the maximum possible current true time is t_j:

$$i_{\max} = floor\left(\frac{t_j - T_0}{T_{int}}\right) \tag{2}$$

Where T_0 is the time of transmission of the first key and T_{int} is the time interval between keys.

- The receiver extracts from P_j the index i of the key that should be used to sign the current packet: $i = j + d$, where d is the disclosure delay.
- If $i_{\max} < i$ then the packet is considered safe and can be stored for future authentication.

For GNSS, however, the assumptions are different. Firstly, the receiver-sender two-way communication is not possible, as GNSS provides one-way communication only. Due to this constraint, the GNSS receiver can be more vulnerable to replay attacks: Unless the GNSS receiver has a means to establish trust in the signal and ensure it is not a replay, the receiver cannot automatically resynchronize with it. Secondly, in network-based TESLA, packets are received separately and with potentially different latencies due to the network, while in GNSS each satellite transmits a continuous

data stream synchronized with the satellite's atomic clock, with only a slow-varying drift due to the satellite-to-receiver varying distance. Finally, instead of a relative synchronization between server and receiver, in GNSS, the server (i.e. the navigation satellite system) is providing the true time, so in the absence of a spoofed signal the receiver is receiving a true time reference.

Synchronization Parameters for GNSS-TESLA

Let us assume the following nomenclature for GNSS-TESLA synchronization:

- *S*: synchronization event at the receiver allowing bootstrapping the GNSS-TESLA protocol.
- t_{true}, t_{sig}, and t_{ref}: time scales for the true time, received time from the GNSS signal, and receiver independent reference time, respectively.
- t_{true}^{S}, t_{sig}^{S}, and t_{ref}^{S}: Event *S* timestamped in the timescales of the true time, GNSS signal time, and GNSS independent reference time, respectively. t_{sig}^{S} corresponds to t_{true}^{S} if the signal comes from the GNSS, but it can be different if it does not. t_{ref}^{S} is equivalent to T in the TESLA description above from [6].
- T_L: loose time synchronization requirement for a secure GNSS-TESLA protocol initialization. It is assumed to be known a-priori by the receiver. It is the delay between MAC and related key disclosure.
- *B*: bound of the t_{ref}^{S} error.

In relation to the "safe packet test" described in [6], T is equivalent to t_{ref}^{S}, D(T) is equivalent to *B*, *d* is equivalent to T_L, and t_j is equivalent to $\hat{t}_{true,W}^{S}$, according to our nomenclature described later in the paper. While the process described here is somewhat different, it follows the same principles as that of [6].

In order to justify the need for *B,* let us assume that at startup the receiver just checks that $t_{ref}^{S} - t_{sig}^{S} < T_L$. This means that if the signal time t_{sig}^{S} is lagging with a delay higher than T_L with respect to the reference time t_{ref}^{S}, the signal is considered a replay, and otherwise it is considered as valid. However, this inequality is considered an over-simplification for GNSS-TESLA during a secure startup, and is therefore invalid. If t_{ref}^{S} has no error, then $t_{ref}^{S} = t_{true}^{S}$. In this case, if the receiver is acquiring the authentic signal, then $t_{sig}^{S} = t_{true}^{S}$ as well, and the receiver can start up independently of T_L. If there is an error uncertainty associated with t_{ref}^{S}, fulfilling the inequality does not imply the fulfilment of the loose time synchronization requirement, as both t_{ref}^{S} and t_{sig}^{S} may be wrong by an unknown amount, which may be greater than T_L.

For the rest of this paper, we assume that there is an error uncertainty associated with t_{ref}^{S}, and is represented by a bound *B*. t_{true}^{S} shall be within the confidence interval $[t_{ref}^{S} - B, t_{ref}^{S} + B]$[1]. We also assume that the receiver knows both t_{ref}^{S} and *B* at startup.

The way to obtain and characterize *B* at startup is out of scope of this paper. It can be based on the statistical characterization of the clock drift since the previous re-synchronization, that gives a certain

[1] Or out of this confidence interval with a probability that is considered so low that the startup is considered secure

probability that the error is contained within the confidence interval, or by the error associated with an external synchronization system that is used at startup, just to name some examples.

GNSS-TESLA synchronization requirements

As a data broadcast authentication protocol, the primary objective of GNSS-TESLA is to authenticate the GNSS data. In order to do so, the receiver has to ensure that the received signal is not delayed by more than T_L, as otherwise an adversary could forge the data and MACs once the key is available. Therefore, the following precondition must hold true to avoid data forging attacks:

$$t^S_{true} - t^S_{sig} < T_L \tag{3}$$

The receiver can receive t^S_{sig} but it does not know t^S_{true}, only t^S_{ref}. It has to make a hypothesis on the value of t^S_{true} based on t^S_{ref} and B. In the following, we take as a reference a worst-case scenario, whereby our estimation of t^S_{true} takes the worst possible value within the confidence interval. The worst possible value is such that t^S_{ref} is delayed as much as possible with respect to t^S_{true}, as this maximizes the success probability of a delay attack. This condition is fulfilled if t^S_{true} had the maximum possible value of the interval. We therefore take an estimated true time value, for this worst-case scenario, as follows:

$$\hat{t}^S_{true,W} = t^S_{ref} + B \tag{4}$$

From (3) and (4) we can define the following precondition:

$$t^S_{ref} - t^S_{sig} < T_L - B \tag{5}$$

Figure 1 illustrates how $\hat{t}^S_{true,W}$ is obtained, and the application of T_L from it to determine the region free of data forging. If t^S_{sig} is at the left of the T_L interval, the data may be forged.

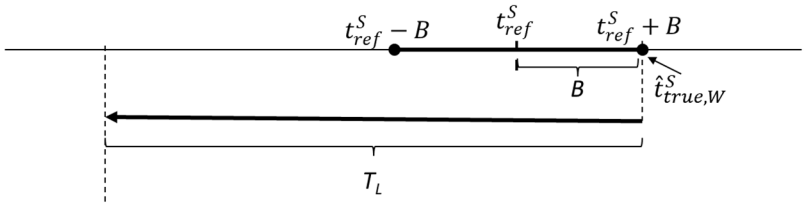

Figure 1 – representation of the receiver reference time, the reference time bound, the estimated true time for the worst-case scenario, and the loose time required for startup

If B is so high that the whole confidence interval ($2B$) is greater than T_L, there will be a region of service unavailability, or potential false alerts if the service is provided. These false alerts will occur when t^S_{ref} runs too fast and t^S_{true} is in the low end of the interval. This is shown in Figure 2. In order to avoid false alerts, the following precondition must hold:

$$B < \frac{T_L}{2} \tag{6}$$

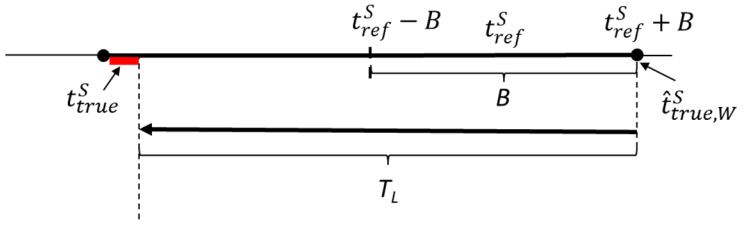

Figure 2 – False alert if B is too high and t_{ref} is advanced with respect to t_{true}

In summary, the three preconditions required for GNSS-TESLA initialization are:

- *Bounding* precondition: The reference time confidence bound *B* must bound the error of t^S_{ref}.
- *Synchronization* precondition: The time difference of the worst-case estimated true time minus the signal time must be below T_L (5).
- *False alert* precondition: To prevent false alerts, the reference time confidence bound *B* must be below *half* of T_L (6).

In the following, we analyze how the receiver must treat different cases of t^S_{sig} with respect to t^S_{ref}. We use Figure 3, which is equivalent to Figure 1 and fulfils (6).

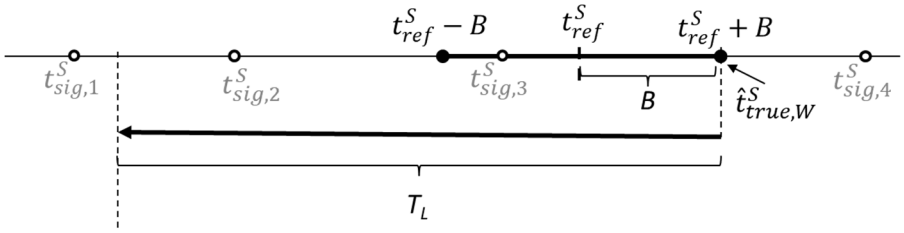

Figure 3 – startup process with different signal times

Case 1) $t^S_{sig,1} < \hat{t}^S_{true,W} - T_L$: The received signal may be a replay with forged data. The receiver must stop and raise an alert.

Case 2) $\hat{t}^S_{true,W} - T_L < t^S_{sig,2} < t^S_{ref} - B$: The received signal cannot be a replay with forged data, but may be a replay, as it is out of the reference time boundaries. The receiver may stop and raise an alert. It may also elect to proceed after raising the alert should the authentication otherwise pass, depending on the use case and application criticality.

Case 3) $t^S_{ref} - B < t^S_{sig,3} < t^S_{ref} + B$: The received signal is within the boundaries of the reference time. The startup process can continue.

Case 4) $t^S_{ref} + B < t^S_{sig,4}$: The received signal is in advance with respect to the reference time. It cannot therefore be a replay with forged data, but it is out of the reference time boundaries. The signal may be an advanced replica, and if processed, the authentication verification would fail. It is also possible that the previous calibration of the reference time was inaccurate or corrupted. The receiver should raise an alert. It may also elect to potentially proceed should the authentication otherwise pass.

Therefore, for a more comprehensive startup process, that includes not only data forging, but also the detection of unauthentic signals, we may consider a fourth precondition:

- *Signal replay* precondition: the difference between the reference time t_{ref}^{S} and the signal time t_{sig}^{S} must be below the reference time confidence bound *B (7).*

$$\left| t_{ref}^{S} - t_{sig}^{S} \right| < B \tag{7}$$

Analysis of False Alerts and Missed Detections

We can reformulate (5) into the equivalent expression (8) as follows:

$$\begin{gathered} t_{ref}^{S} - t_{sig}^{S} < T_L - B \\ (t_{ref}^{S} - t_{true}^{S}) - (t_{sig}^{S} - t_{true}^{S}) < T_L - B \\ b_{ref} - b_{sig} < T_L - B \end{gathered} \tag{8}$$

Where b_{ref} and b_{sig} are the time biases with respect to the true time. Note also that the bounding precondition can be expressed as follows*:*

$$-B < b_{ref} < B \tag{9}$$

We analyze two hypotheses: the hypothesis of no attack (H_0), and the hypothesis of an attack leading to data forging at the minimum required delay (H_1). Table 1 presents how they affect the bounding and synchronization preconditions. In the case of the null hypothesis H_0, as the true signal is received, there is no signal offset, so b_{sig} = 0 . This means that the synchronization precondition to fulfil is $b_{ref} < T_L - B$. When the receiver is under attack (H_1), we assume that the signal is delayed by the minimum necessary to forge the data, so b_{sig} = -T_L, and the synchronization precondition to fulfil becomes $b_{ref} < -B$. However, the bounding and synchronization preconditions cannot be fulfilled simultaneously, and therefore there cannot be missed detections.

Hypothesis	Bounding Precondition (9)	Synchronization Precondition (8)
Receiver operates under null hypothesis (no attack) H_0: b_{sig} = 0	$-B < b_{ref} < B$	$b_{ref} < T_L - B$
Receiver operates under alterative hypothesis (attack) H_1: b_{sig} = -T_L	$-B < b_{ref} < B$	$b_{ref} < -B$

Table 1 – Hypothesis testing of no attack (H_0) and attack with the minimum delay required for data forging (H_1), including precondition 1, to be fulfilled to continue startup process, and precondition 2, fulfilled by the definition of B.

Figure 4 and Figure 5 show the result of the bounding (9) and synchronization (8) preconditions for different values of b_{ref}, where Figure 4 represents the null hypothesis and Figure 5 the alternative one. The figures assume that there is no resynchronization and *B* grows linearly over time since the last calibration.

We can see in Figure 4 that the false-alert zone starts after $B > T_L/2$, as anticipated by the false alert precondition (6). This is depicted by the triangular orange zone in the figure: while $B < T_L/2$, we can ensure no false alerts, but after that, the possible inaccuracy of b_{ref} will often lead to the case depicted in Figure 2. Due to this reason, the recommended operational zone, colored in green, stops when $B = T_L/2$. The light grey areas represent when the bounding precondition is not fulfilled, and the dark grey areas when neither the bounding nor the synchronization preconditions are fulfilled.

Figure 4 also shows three sample cases, *C1, C2*, and *C3*, to better illustrate the precondition checks. In *C1*, where $B < T_L/2$, if the bounding precondition (8) is fulfilled, there will be no false alerts, whatever is the value of b_{ref}. In *C2*, as B > $T_L/2$, the synchronization precondition (9) is not fulfilled for high

values of b_{ref}, leading to a false alert. This effect is aggravated in *C3*, where $B > T_L$. The main problem of exceeding $B > T_L$ is that, even if b_{ref} is small and therefore the reference time is around the true time, a false alert will be raised. Note that the fact that the bounding and synchronization preconditions will be sometimes fulfilled beyond $B > T_L$, for negative b_{ref} values, does not imply that signals replayed with a delay beyond T_L will go undetected, as shown in Figure 5.

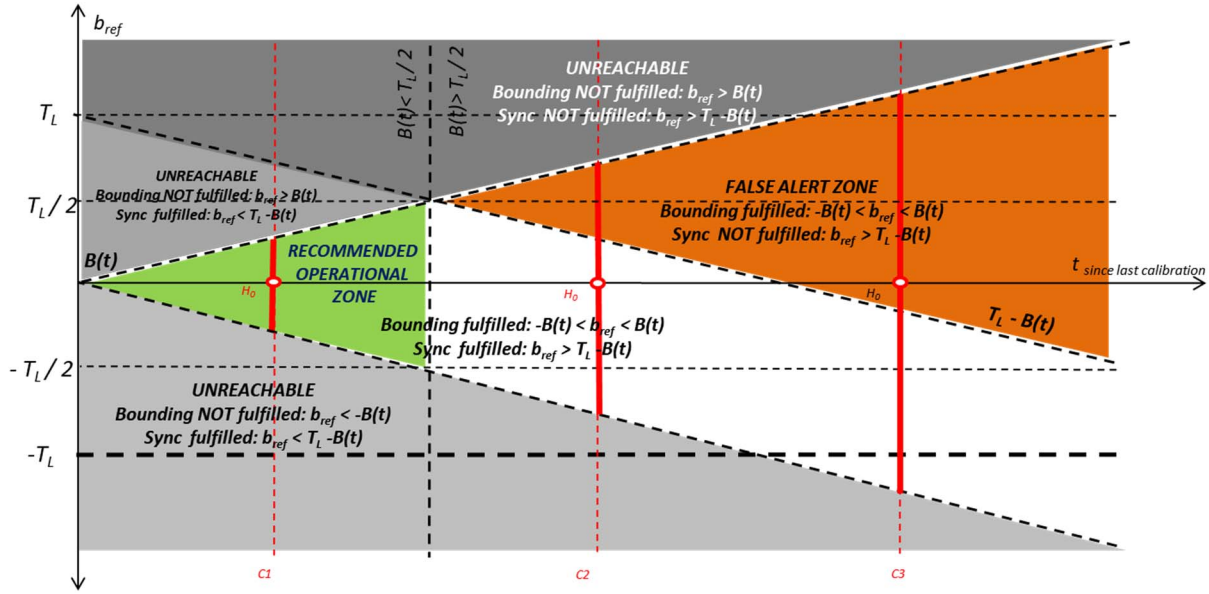

Figure 4 – Detection regions over time since last calibration, in case of no attack, H_0: b_{sig} = 0.

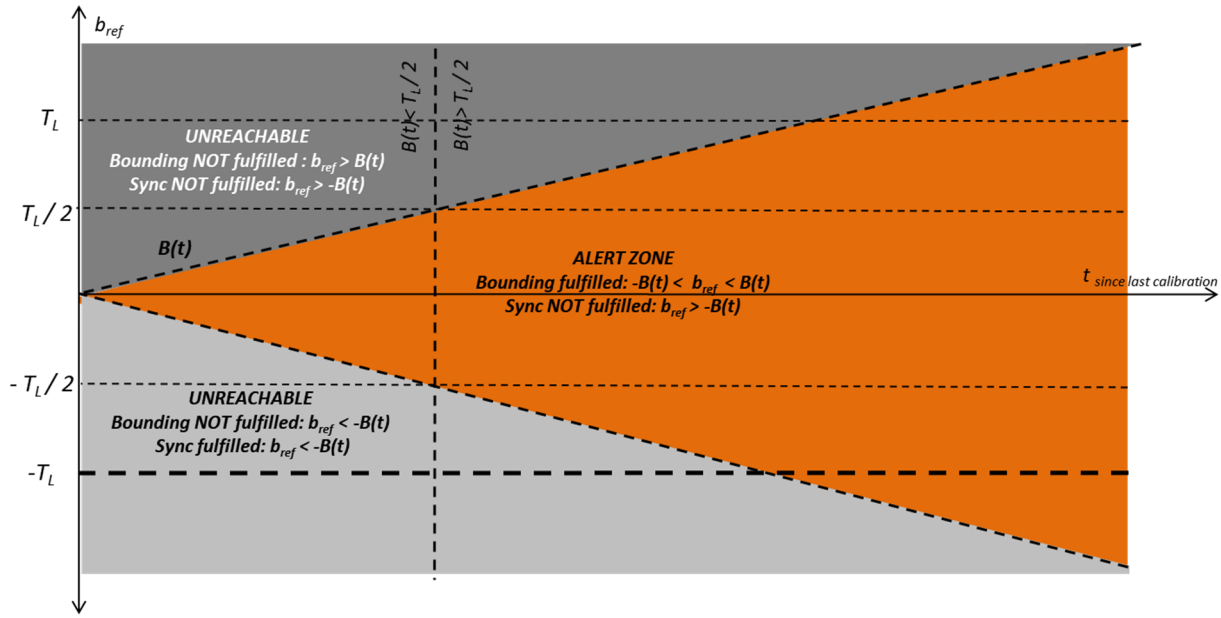

Figure 5 – Detection regions over time since last calibration, in case of data forging attack, H_1: b_{sig} = -T_L. As both preconditions can't hold simultaneously irrespectively of b_{ref}, attacks are detected.

Figure 5 shows the alternative hypothesis H_1 where the signal is delayed with b_{sig} = -T_L, which is the minimum delay to forge the data. In this case, the check will only pass if $b_{ref} < -B$. However, this condition will never be fulfilled due the bounding precondition, so if the signal is data-spoofed, the process will *always* detect it, as long as *B* effectively bounds the reference time error. This is true also of $B > T_L$.

GNSS-TESLA Synchronization Procedures

This section presents two proposals for synchronization startup. The proposals are based on the abovementioned three preconditions (bounding, synchronization, and false alerts), with some small variants. The first method for implementing the synchronization logic during GNSS-TESLA receiver startup is depicted in Figure 6. It assumes that the receiver can periodically re-calibrate its internal clock, and is based on the following steps:

1. At startup, get t_{ref}^{S} and $B(t_{ref}^{S} - t_{last\_calibration})$ from a reference receiver Real Time Clock (RTC) (e.g. [7] [8]).
2. Get t_{sig}^{S} from the signal in space, by calculating a GNSS position and time fix, still to be authenticated.
3. Lookup the appropriate value of T_L for the TESLA protocol.
4. Check that the internal reference time confidence bound B is within limits ($T_L/2$). If not, raise an alert to the operator to initiate an alternate synchronization procedure.
5. Check that $t_{ref}^{S} - t_{sig}^{S} < T_L - B$. If this is the case, continue startup.
6. If $t_{ref}^{S} - t_{sig}^{S} \geq T_L - B$, raise an alert to a potential spoofing attack.
7. Check the validity of the TESLA keys and MACS, if they are valid then proceed as normal and collect new data to calibrate the RTC (i.e. $t_{ref}^{S} - t_{sig}^{S}$ data presuming that $t_{sig}^{S} = t_{true}^{S}$, under the more detailed assumptions below).

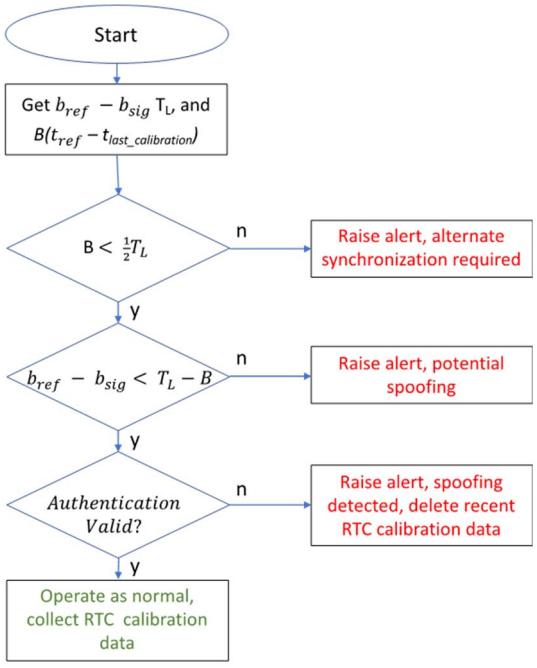

Figure 6 – GNSS-TESLA synchronization startup flow diagram – Method 1

Another similar synchronization procedure for GNSS-TESLA receiver startup is depicted in Figure 7. It is based on the following steps:

1. At startup, get t_{ref}^{S} and $B(t_{ref}^{S} - t_{last\_calibration})$ from the reference time source (receiver real time clock, external source, or other).
2. Get t_{sig}^{S} from the signal in space, by calculating a GNSS position and time fix, still to be authenticated.

3. Lookup the appropriate value of T_L for the TESLA protocol (the calculation of T_L is out of the scope of this paper).
4. Check that the internal reference time confidence bound *B* is within limits (T_L / 2). If not, re-synchronize through an alternate synchronization procedure and go to step 1 if possible.
5. Check that the difference between t_{sig}^S and t_{ref}^S is within t_{ref}^S boundaries defined by *B* (*signal replay precondition*).
6. If t_{sig}^S and t_{ref}^S differ beyond the expectation, check if a signal replay with data forging attack is possible (*synchronization precondition*). If this is the case, report a possible data forging alert. If not, report a signal anomaly (unauthentic signal).
7. If the result of 5. is positive, as this precondition is more restrictive than the *synchronization precondition*, consider the GNSS data trustable and continue authenticating the data. If the authentication is valid (i.e. it passes the authentication verification), the position can be considered as data-authenticated. Otherwise raise an authentication alert.

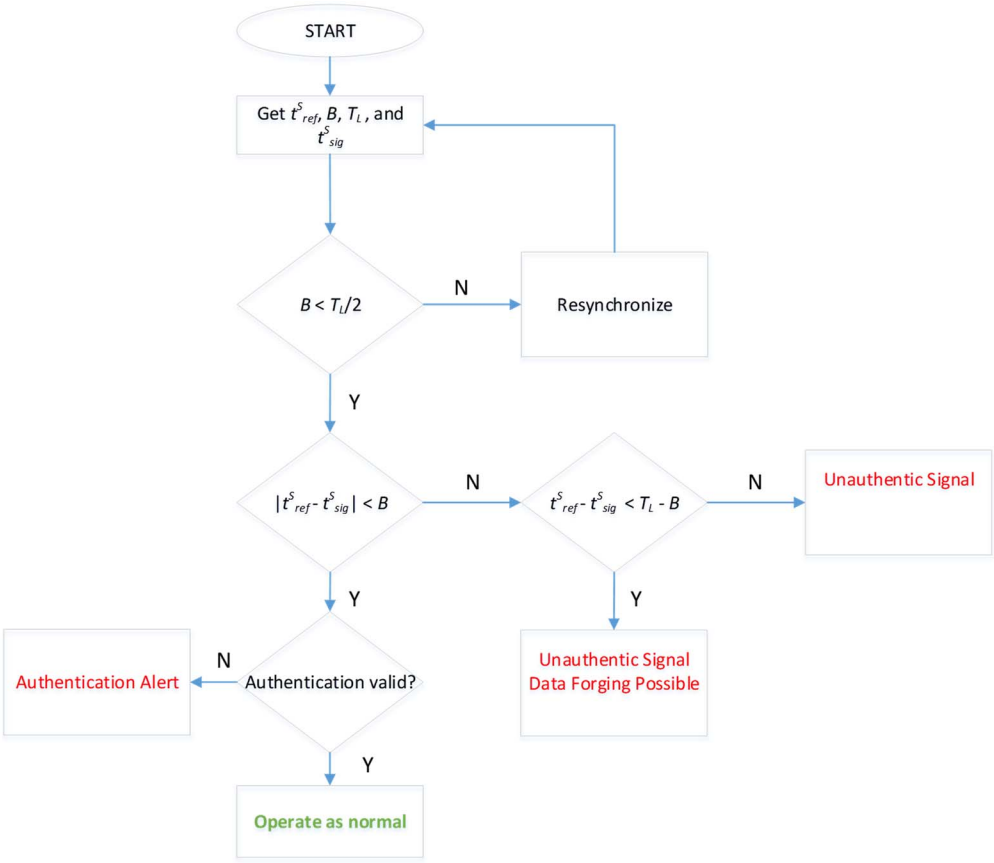

Figure 7 – GNSS-TESLA synchronization startup flow diagram - – Method 2

The second procedure slightly differs from the first one mainly in that it incorporates the signal replay precondition. Also, the management of unfulfilled preconditions is slightly different. In summary, these are just examples that show that, depending on the security profile of the application, some variants in the startup procedure can be introduced. We finalise this section with some further considerations on signal synchronization and RTC calibration.

Signal synchronization: The above procedures assume that the t_{sig}^S is obtained once the SIS of at least four satellites is acquired and tracked and a still-unauthenticated position and time solution is

calculated. However, the receiver could also obtain the signal time from a single satellite, at the expense of an uncertainty of some milliseconds to be accounted for in the process.

Real-time clocks and their calibration: A real-time clock is an independent clock that keeps time even when the receiver is powered off. In this concept it will need to be calibrated relative to GNSS time. This should be done very carefully and only using data known to be valid. Initially, this calibration may be performed by the receiver manufacturer. In the Method 1 described above, two different calibration approaches are used: An initial calibration in a trusted environment (i.e. where spoofing is not observed and trusted not to be present) or against an independent and trusted time source. These approaches are beyond the scope of this paper and will not be further described. Another approach is when B is consistently below $T_L/2$, then the calibration can be updated when all of the above conditions are met. For example, an RTC may have an accuracy bound of five parts per million (ppm) or to within five seconds after one million seconds have elapsed. Thus if we knew t_{ref}^{S} accurately one day (86,400 seconds) ago, then we would have a value for B of 0.432 seconds after being turned off for one day. Assuming we follow our above procedure, once we passed all of our checks and authenticated the signal, we could collect and store values of $t_{ref}^{S} - t_{sig}^{S}$. These would be used to calibrate t_{ref}^{S} the next time it is called upon at start up, assuming, as abovementioned, that the receiver places a certain confidence on the authenticity of t_{sig}^{S}. Further information and characterization of RTCs can be found in [9].

Authenticated Network Time Synchronization

This section and the following ones deal with practical aspects of GNSS receiver synchronization for different applications, starting with assisted receivers who can benefit from authenticated network time synchronization. In the event that the user clock uncertainty has grown beyond the secure time synchronization requirement, then a re-synchronization of the RTC will be required. One way to achieve this is through an authenticated network time transfer. As described in [11], two-way communication is a necessary condition for secure time transfer. This is the only mechanism to avoid a simple replay attack. The time-seeker constructs a message requesting a time response from a time server, including a “nonce”, or cryptographically secure random number. The server responds with a time estimate and a digital signature, or message authentication code, that includes both the server’s timestamp and the client-generated nonce. In this way a causal link is established between the request and the response. The client measures the round-trip time between client and server and generates a best estimate of the current local time as the server time plus one half the round trip time. An implementation of this procedure is also described in [5] for the TESLA protocol, as abovementioned.

This approach remains vulnerable to a Man-in-the-Middle (MitM) attack, in which a malicious adversary positions themselves on the network between the client and server and selectively delays incoming and outgoing packets. For example, in the extreme case where the client and server are co-located, the true round trip time should be zero, however the attacker can deliver the request packets to the server with no delay, and the response packets to the client with a delay of, say, one second, thereby yielding a client clock error of 0.5 seconds early. On the other hand the attacker can delay the request packets and deliver the response packets promptly thereby yielding a client clock error of 0.5 seconds late. The maximum attacker induced time error is +/- RTT/2, where RTT is the round trip time.

Figure 8 shows the principle of operation of all network based time synchronization mechanisms. Here time flows downwards, with the client timeline on the left and the server on the right. The nominal operating condition is shown in the left plot, while the condition under a MitM attack is shown on the right.

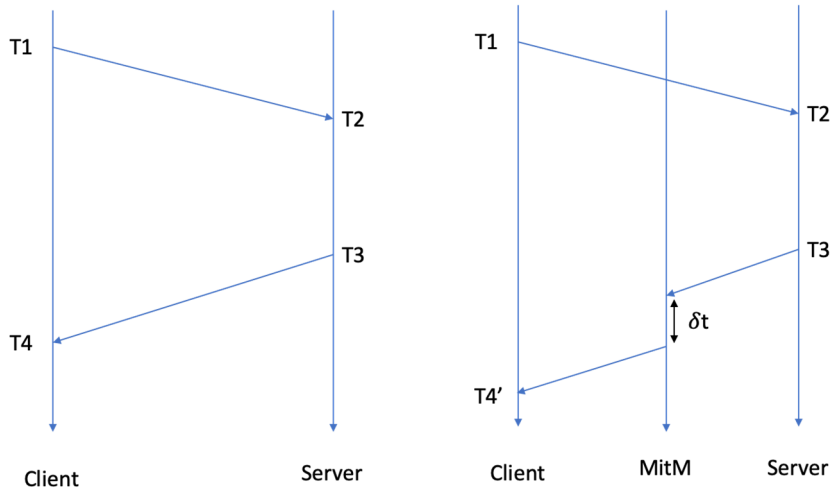

Figure 8 - Network time synchronization. Left: no attack; Right: Man-in-the-Middle attack. Note the extra delay δt introduced by the MitM, this translates into a clock error of $\delta t/2$.

The Network Time Protocol (NTP) is one of the most commonly used network time synchronization protocols, but while a number of attempts have been made to introduce cryptographic authentication it remains largely used without any security features. The latest effort to address this at the standards level is the Network Time Security (NTS) over NTP protocol, which is currently at the draft stage with the Internet Engineering Task Force (IETF) [12] [13]. This protocol is essentially as described above, but the individual time offsets from multiple servers are aggregated and filtered over long time periods. In general NTP is designed to run continuously providing accurate and stable time to network connected devices.

An alternative authenticated network synchronization protocol has recently been proposed by Google. This protocol is called Roughtime and is based on a snapshot approach to synchronization with, as the name suggests, a relatively low accuracy, but with very high reliability. Again the time synchronization is very similar to that described above, but Roughtime utilises multiple servers in a chain configuration. The protocol specifically allows the detection of malicious spoofers and their elimination from the time transfer computation. This protocol is also currently undergoing review in the IETF standardization process [12] [13].

Both NTS and Roughtime rely on standard network communications protocols, and as such the uncertainty bound on the transferred time is constrained to one half of the measured round trip time, due to the vulnerability to MitM attacks described above. With a good geographic spread of servers, round trip times of the order of 10's to 100's of milliseconds can be expected given a reasonably good, low latency network connection, such as ethernet or 3G data connections. Thus either of these two protocols, or indeed any other similar protocol, are viable candidates for the resynchronization of the reference clock, provided that a reliable network connection is periodically available at a rate commensurate with the resynchronization needs of the RTC.

Application to Aviation

The above approach is very well suited for commercial aviation as these aircraft rarely go 24 hours without being powered on and they move from one location to another. Therefore, it is highly unlikely that they will be subjected to spoofing for extended periods of time. A successful attack on our synchronization approach would be to echo valid signals, but to slowly drive the receiver time backwards until it is more than T_L behind the true time. As a result, T_L should be chosen such that this requires many days to succeed. For example, again using 5 ppm as the specification on the RTC along with a value of 6 seconds for T_L would require that a spoofer spend at least more than a week slowly offsetting the signal before they can successfully offset the receiver clock sufficiently. Unfortunately, general aviation aircraft (as well as many other users) may indeed go more than a week without operating their receivers and updating their calibration. Here we would require either a tighter specification on the RTC (e.g. 1 ppm), a larger T_L , or an alternative calibration method.

One possible solution is to use a method of multiple instances where different MACs are sent with different T_L values. For example, one MAC can use keys that are released 6 seconds later while another MAC is also transmitted, but this MAC uses a key that will be released 120, or 300 seconds later, as implemented e.g. for Galileo OSNMA with the *slow MAC* concept [10]. An advantage of this approach is that that the same TESLA keychain may be used for both MACs. Thus, the required number of bits to be transmitted is minimized as a MAC may be only 30 bits [3]. These slow MACs also do not need to be transmitted nearly as often as the fast MACs. Following our same example, if the user has not turned on their receiver in the last two weeks, but has turned it on in the last forty, then they will have to wait at least 120 seconds to validate the slow MAC but then they will have been able to verify that the received data has not been forged and they can now use the received signal to update the calibration on their RTC, under the assumption that the signal is trusted. Further, with a TL of two minutes, most operators should have access to an independent time source with greater accuracy (e.g. wrist watch, laptop with NTP, etc.).

Conclusions and Future Work

Independent loose time synchronization of GNSS receivers from an external time reference of an accuracy of some seconds or minutes is required for GNSS time-delayed authentication protocols, be it for data or spreading code authentication. It may also be a useful feature to protect in general against spoofing attacks.

This paper looks at independent time synchronization processes particularized for the TESLA protocol applied to GNSS, establishing the necessary and recommended preconditions. The GNSS receiver must provide, at the synchronization event *S*, its reference time t_{ref}^{S} and a bound *B* of its error (*bounding precondition)*, and check that t_{ref}^{S} and the time from the GNSS signal t_{sig}^{S} comply with this inequality: $t_{ref}^{S} - t_{sig}^{S} < T_L - B$ *(synchronization precondition)*. The receiver must also re-synchronize before *B* exceeds $T_L/2$ to avoid false alerts (*false alert precondition*). If *B* exceeds this limit, data forging still will be detected, although with an increasing false alert rate. In addition, if the difference between t_{ref}^{S} and t_{sig}^{S} is above *B*, the signal can be considered unauthentic (*signal replay precondition*).

Secure GNSS-TESLA synchronization procedures at startup based on these rules are proposed, with some considerations about RTC recalibration, in case an RTC is used as the external time reference.

The paper also qualitatively analyses external synchronization through secure time transfer from assistance networks, and the potential implementation of such procedures in the domain of aviation.

Future work on the topic includes defining a logic when *B* exceeds $T_L/2$, depending on whether the use case can tolerate false alerts, and in that case, how far the receiver could exceed that region depending on the tolerable false alert rate. In addition to this, the bound *B* was assumed to increase monotonically and linearly, but in the future this bound, its evolution, and its integrity level will need to be properly characterized. Other remaining work include the recalibration procedures, including conditions on which the receiver can be recalibrated (i.e. the bound *B* can be reset to zero or reduced), and the development and experimentation of external synchronization procedures based on real receivers and RTCs.

References

[1] Anderson, Jon M.; Carroll, Katherine L.; et al., "Chips-Message Robust Authentication (Chimera) for GPS Civilian Signals," in *ION GNSS+*, Portland, OR, 2017.

[2] L. Scott, "Anti-Spoofing & Authenticated Signal Architectures for Civil Navigation Systems," *ION GPS,* 2003.

[3] A. Neish, T. Walter and J. D. Powell, "SBAS Data Authentication: A Concept of Operations," in *ION GNSS+ 2019*, Miami, FL, 2019.

[4] I. Fernandez-Hernandez, V. Rijmen, G. Seco-Granados, J. Simon, I. Rodriguez and J. D. Calle, "A Navigation Message Authentication Proposal for the Galileo Open Service," *Journal of the Insitute of Navigation,* no. Spring, pp. pp. 85-102, 2016.

[5] A. Perrig, R. Canetti, J. D. Tygar and D. Song, "The TESLA Broadcast Authentication Protocol," *CryptoBytes,* 2002.

[6] V. Roca, A. Francillon and S. Faurite, "RFC 5776 - Use of Timed Efficient Stream Loss-Tolerant Authentication (TESLA) in the Asynchronous Layered Coding (ALC) and NACK-Oriented Reliable Multicast (NORM) Protocols," Internet Engineering Task Force (IETF), 2010.

[7] "STMicroelectronics," [Online]. Available: https://www.st.com/en/clocks-and-timers/real-time-clock-rtc-ics.html. [Accessed Dec 2019].

[8] Micro Crystal, [Online]. Available: https://www.microcrystal.com/en/products/real-time-clock-rtc/. [Accessed Dec 2019].

[9] SEIKO EPSON CORPORATION, "High-Stability Time Adjustment with Real-Time Clock Module - White paper," 2014.

[10] I. Fernández, V. Rijmen, T. Ashur, P. Walker, G. Seco, J. Simón, C. Sarto, D. Burkey and O. Pozzobon., "Galileo Navigation Message Authentication Specification for Signal-In-Space Testing – v1.0," European Commission, 2016.

[11] L. Narula and T. E. Humphreys, "Requirements for secure clock synchronization," *IEEE Journal of Selected Topics in Signal Processing ,* vol. 12, no. 4, p. pp. 749–762., 2018.

[12] D. Franke, D. Sibold, K. Teichel, M. Dansarie and R. Sundblad, "Network Time Security for the Network Time Protocol. Internet-Draft.," IETF Secretariat, url: http://www.ietf.org/internet-drafts/draft-ietf-ntp-using-nts-for-ntp-20.txt, July 2019.

[13] A. Malhotra, A. Langley and W. Ladd, "Roughtime. Internet-Draft," IETF Secretariat, http://www.ietf.org/internet-drafts/draft-roughtime-aanchal-03.txt, July 2019.