Experimental Evaluation of Attacks on TESLA-secured Time Synchronization Protocols

K. Teichel, G. Hildermeier

Acknowledgement: Parts of the presented work are used in the project 17IND06 (FutureGrid II) which has received funding from the EMPIR programme co-financed by the Participating States and from the European Union's Horizon 2020 research and innovation programme.

Experimental Evaluation of Attacks on TESLA-secured Time Synchronization Protocols

Kristof Teichel[1]✉ and Gregor Hildermeier[2]

[1] Physikalisch-Technische Bundesanstalt,
Bundesallee 100, 38116 Braunschweig, Germany
kristof.teichel@ptb.de
[2] Technische Universität Braunschweig,
38092 Braunschweig, Germany

Abstract. There is an increasingly relevant class of protocols that employ TESLA stream authentication to provide authenticity for one-way time synchronization. For such protocols, an interdependency between synchronization and security has been found to theoretically enable attackers to render the security measures useless. We evaluate to what extent this attack works in practice. To this end, we use a tailor-made configurable testbed implementation to simulate behaviors of TESLA-protected one-way synchronization protocols in hostile networks. In particular, this lets us confirm vulnerabilities to the attack for two published protocols, TinySeRSync and ASTS. Our analysis also yields a set of countermeasures, with which in-development and future specifications can potentially use TESLA to successfully secure one-way time synchronization.

Keywords: (One-Way) Time Synchronization Protocols · TESLA · Authentication · Experimental Attack Analysis · ASTS · TinySeRSync

1 Introduction

Time synchronization has been a crucial mechanism since the creation of the first computer networks. In many settings, clock synchronization failure or manipulation is unacceptable. With an increasing focus on security in recent years, the demand for authenticity in time synchronization protocol specifications has increased. A mechanism called "Timed Efficient Stream Loss-tolerant Authentication" (TESLA) [10, 11], which employs delayed key disclosure, has been adopted for providing authenticity for one-way time synchronization protocols. TESLA is attractive because it meets a combined challenge: it supplies well-scaling authentication in systems with large numbers of slaves and mostly one-way communication flow, but at the same time offers the speed of symmetric cryptography. However, a potential attack vector has been discovered in which, under certain circumstances, the authenticity provided by TESLA can be compromised entirely when it is used to protect one-way time synchronization [16]. Our main focus in this paper is the question to what extent this attack is relevant in practice, on standardized protocol specifications.

At least two published protocols have already made use of TESLA-based authentication mechanisms to secure time-related messages: the "Secure and Resilient Time Synchronization protocol" [15] (TinySeRSync) and the "Agile Secure Time Synchronization protocol" (ASTS) [17], both of which were developed for wireless sensor networks. Furthermore, the network time synchronization community is reviewing how to apply TESLA in ongoing specification work. The Institute of Electrical and Electronics Engineers describes a TESLA-based security scheme in the upcoming version of its Precision Time Protocol [1] specification. The Internet Engineering Task Force has been discussing TESLA's use for the broadcast mode of the Network Time Protocol [9] in the context of the ongoing Network Time Security specification [13, 14]. The European Space Agency is building TESLA protection into the Open Service Message Authentication scheme [5] of Galileo, the European global Navigation satellite system. Thus, depending on how these specifications turn out, TESLA-protected one-way synchronization might in the future be employed on billions of devices.

The complexitiy that arises from the combination of TESLA with time synchronization procedures makes evaluating the susceptibility of a given protocol to the attack a hard problem. It is especially difficult to prove that a protocol is positively secure, since this requires proof of absence of a successful attack path in a highly complex system. Proving that a protocol is not secure is simple: one only has to provide an attack scenario as a witness. However, it is not necessarily easy, since finding and documenting such a scenario requires a solid understanding of the mechanisms involved and a convenient way to log the events that lead to it.

The treatment in [16] is the only in-depth analysis on the problem that we could find. However, it has limitations that we wanted to expand upon. First, the analysis is limited to a generic, generalized and abstracted protocol representing the class of all TESLA-protected one-way synchronization protocols without regard for any potential specific intricacies of each. Second, the analysis is executed only with model checking via the UPPAAL model checker, so any abstractions made in the creation of the model might influence the applicability of the results. Third, the model and the tool do not allow for even remotely realistic ratios of interval lengths to time units: in the UPPAAL model, an interval can only be in the order of 10 time units long before state space size exceeds memory limitations. Other analysis [2, 3] avoids the details of the issue altoghether, generally noting that the interdependency of timing and security is a potentially critical problem.

In this paper, we make three key contributions to the subject area of practical vulnerability to the attack of existing and future protocol specifications. We use our own configurable testbed implementation to run simulations of TinySeRSync and ASTS in which they are attacked by an adversary using techniques from [16]. The data from these simulations enables us to make well-founded statements regarding the existing protocols. First, a faithful implementation of TinySeRSync without further mitigation is attackable, albeit with a much easier attack scheme than the one originally outlined. Second, a faithful implementation of ASTS without further mitigation is attackable with exactly the methods outlined in [16].

These statements about the established protocol definition constitute our first key contribution to the subject area. They represent an improvement in the attack analysis, from conjectured vulnerability of concrete protocols after loose study of the describing papers, towards field tests on physically distributed devices running faithful implementations of the protocol descriptions. Our testbed implementation directly represents our second contribution, since it is generic enough that it can be adapted to simulate both existing and future protocols and their behavior, especially with regards to the attack within the scope of this work. Furthermore, we deduce a set of countermeasures to mitigate the attack. These, combined with the analysis and derivation that yields them and puts them into practical context (an improvement over countermeasure suggestions in [16]), constitute our third key contribution. The given countermeasures can be included in future specifications of protocols which involve TESLA-protected one-way time synchronization. We feel that the number of such specifications currently in development gives our latter contributions extra weight.

This paper is structured as follows: Section 2 provides an introduction to basic time synchronization techniques, to the TESLA protocol and to the attack vector on which we base our simulation and analysis. In Section 3, we present the setup of our experiments and give an overview of our implementation. Section 4 discusses the results of our evaluation. This comprises an overview of our analysis regarding protocol vulnerability as well as a derivation and evaluation of countermeasures. Section 5 concludes this paper.

2 Preliminaries

Before we delve into details about results and interpretation of our simulation runs, we use this section to introduce protocols and techniques that are essential to this work, as well as a rough description of the relevant attack vector.

2.1 One-Way and Two-Way Time Synchronization

Generally, network-based time synchronization is achieved in one of two ways: with one-way or with two-way communication. In two-way mode, the participants exchange time request and response messages, as depicted in Fig. 1 (left). By calulations under the assumption of symmetric network delays (i.e. $\epsilon = 0.5$ in the figure), the client can obtain a value for the clock offset with a maximum error of half the network round-trip [8] (i. e. $\delta/2$ in the figure). In one-way mode, a master periodically sends out messages to many slaves, as depicted in Fig. 1 (right). A given slave then needs a good estimate of the transmission time of the packets in order to calculate the clock offset. Note that there are no guarantees for the slave regarding the maximum error of these calculations, and specifically any artificial delay on the transmission packet (P in the figure) adds to that error.

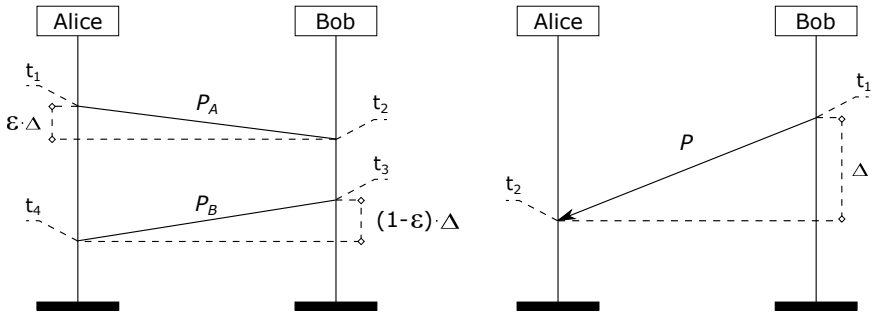


Fig. 1. Two-way (left) and one-way (right) time synchronization (*cf.* [16]).

2.2 Overview of TESLA

The TESLA protocol [10] was originally designed to authenticate media streams, combining the scalability of asymmetric schemes with the high speed of symmetric cryptography. It applies symmetric-key cryptography, but creates asymmetry by making use of time progression: packets are signed and sent, but the key used to sign them is only disclosed after the so-called *disclosure lag*. A received packet is buffered and can be authenticated only after the key is disclosed. As long as the slave can be certain that the key was not disclosed before the packet was received the authenticity of the signature is guaranteed. In order for TESLA to work, two main concepts are deployed:

The first main concept that is employed is that of using a key chain to be able to commit to a key before using it. To generate a key chain, the master applies a one-way pseudo-random function F to a secret K_n to compute a key value $K_{n-1} = F(K_n)$. It repeats this to obtain $K_{n-i} = F^i(K_n)$ up to K_{-1}, which is called the *key chain commit*, while the next key K_0 is the first one to be used. This creates a chain of keys, with each key being verifiable by only its predecessors. The master applies F', another pseudo-random function, to a *key value* K_i in the chain to generate a *key* K'_i. This key K'_i is used to create the MAC for a packet. Before sending any packets, the master commits to the entire chain by publishing K_{-1}, as well as F and F'.

The second main concept is that of using time intervals. The master defines the starting time T_0 and the length T_Δ of a time interval I_i, $i = 0, \ldots, n$. Each interval I_i is associated with the key K_i. Every packet sent in interval I_i is signed with the key K'_i. In interval I_{i+d}, the key value K_i is then disclosed. A depiction of this concept is presented in Fig. 2.

2.3 Attacking TESLA-secured Time Synchronisation Protocols

The attack described in this section was first formulated in [16]. It forms the basis for the behavior of the attacker in our implementation, and is essential throughout this paper. The attack is enabled when the TESLA protocol (or any variant of a scheduled delayed key disclosure) is used to authenticate time

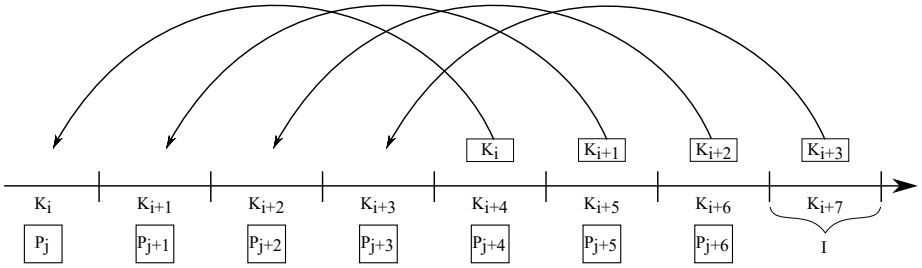


Fig. 2. TESLA authenticates packets with keys disclosed in subsequent packets, in this example after a disclosure delay of $d = 4$ intervals.

distribution in a one-way synchronization setting. Since the protocol itself is time-dependent but, in this application, also directly influences the degree of clock synchronization, the packet authenticity provided can be compromised.

The attacker is assumed to operate under the Dolev-Yao attack model [4]. It is therefore capable of impersonating any member of the network, as long as there is no extra authentication mechanism in place. Notably, it is also capable of delaying any packet. This has a special significance in the context of time synchronization protocols and in the attack explained below.

Attack Synopsis The attack has two phases. *Phase One* has multiple steps:

1. The attacker consistently delays any packets from the master by δ_1.
2. As soon as the slave uses the time data in the first delayed packet to adjust its clock (usually when it verifies the packet after its disclosure lag), the introduced delay δ_1 starts to take effect.
3. This desynchronization increases the time by which the slave will accept a packet as timely. The attacker is thus able to delay packets by $\delta_1 + \delta_2$.
4. The attacker continues to increase the delay according to the step above.

Eventually, the *desynchroization condition* will be fulfilled, meaning that the clocks will have desynchronized by more than $(d-1)T_\Delta$, where d is the disclosure lag. Phase One of the attack is then complete and the attacker can perform *Phase Two* of the attack. It is now possible to intercept any packet from the master, wait until its respective key has been disclosed and replace it with a bogus packet, for which the attacker can generate a correct MAC. The slave's clock is so far behind that it accepts the packet as timely. By this point, the attacker can fully impersonate the master (including the ability to create valid MACs for forged packets), thus breaking the security of the protocol.

Example Attack To further illustrate particularly Phase One of the attack, consider the schematically depiction in Fig. 3, where it is demonstrated with specific values $d = 2$ and $T_\Delta = 6\,\mathrm{s}$ and a delay increment of $4\,\mathrm{s}$ (numbers chosen for comprehensibility and brevity of the scheme).

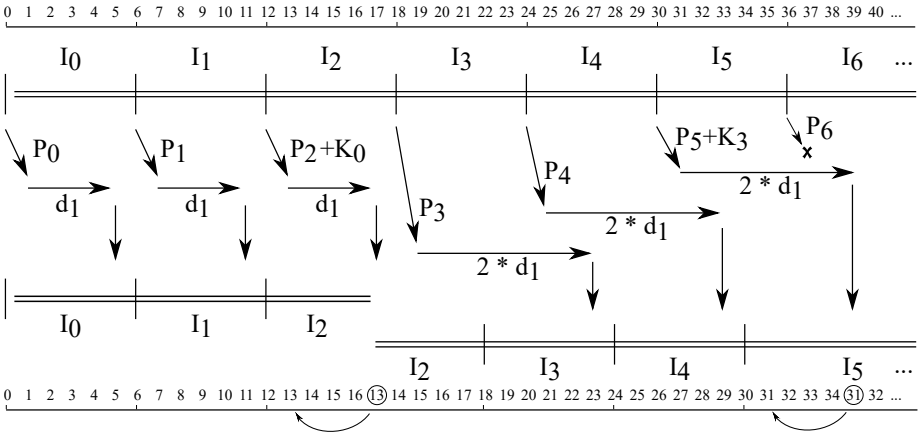


Fig. 3. Illustration of the attack on an example network. Alice adjusts her clock after receiving P_2 and P_5, intervals I_2 and I_5 are prolonged accordingly.

- *Step 1:* The attacker delays P_0 by $\delta_1 = 4\,\text{s}$. The slave checks the timeliness of P_0 and accepts it as timely.
- *Step 2:* Packets P_1 and P_2 are delayed in the same way as P_0. The slave receives P_2 and the key value K_0. The slave authenticates P_0, concludes that its clock is fast by 4 seconds, and adjusts it accordingly.
- *Step 3:* The attacker delays P_3 and subsequent packets, by $\delta_1 + \delta_2 = 8\,\text{s}$.
- *Step 4:* In our example, this increase is sufficient, since $8\,\text{s} > (d-1)T_\Delta$.

The slave's clock synchronization error is already large enough after it receives P_5, which enables it to process P_3, after which it adjusts its clock as though it were fast by another 4 seconds. The slave's clock now reads 31 s, while the master's clock at the same time reads 39 s. The offset between the two clocks has surpassed the value $(d-1)T_\Delta$; hence, the desynchronization condition is satisfied and Phase One is therefore complete and *Phase Two* can now start. The attacker prevents P_6 and the following packets from ever reaching the slave. It waits until the master discloses K_6 in interval I_8 and derives K_6'. It can now forge a packet Q_6 with any bogus time chosen, use K_6' to create a valid MAC and forward it to the slave. In another forged packet Q_8, it includes the original and correct key value K_6. The slave will use this key to (successfully) authenticate the bogus packet Q_6, since K_6 belongs to the key chain. The slave now uses the bogus time data in Q_6 to adjust its clock, and *Phase Two* is therefore successful.

3 Experiment Setup

The overarching goal for this work was to answer the question to what extent the idea for the attack from [16] would work in practice, on implementations of time synchronization protocols running on distributed devices over physical networks.

3.1 Objectives and Approach

In particular, the analysis was desired to be of a nature that would be helpful in the standardization of future protocols securing one-way time synchronization with mechanisms similar to TESLA. This requirement combined with the overall goal yielded a number of specific objectives:

- **Primary objective:** to provide either of the following:
  - Proof that at least one faithful implementation of an exisiting protocol specification was successfully attacked,
  - Statement that no implementation of the candidate protocols could be successfully attacked, with a detailed explanation of why the attack failed.
- **Secondary objective** (in case at least one implementation was successfully attacked): to investigate what role any configurable protocol parameters would play. Specifically, to provide both of the following:
  - A set of parameters under which the attack was successful,
  - A different set of parameters under which the attack was not successful.

Therefore, the two existing finished protocol specifications (TinySeRSync and ASTS) mentioned explicitly in that paper were investigated.

The shortest path to a definitive answer would have been to get an original implementation of either of them running in the lab and perform a successful attack on it. It turned out, however, that we would be unable to go that route. For ASTS, there is to the best of our knowledge simply no publically available implementation. Moreover, while there is code available for TinySeRSync, getting it to run proved too difficult, due to the fact that it was developed for the now unavailable TinyOS 1, and is incompatible with TinyOS 2. We therefore decided to create a new implementation [7] of a highly configurable TESLA-protected one-way synchronization protocol and run our experiments on that. This does represent a caveat with regard to the proof of attackablity of the specific protocols. However, we would like to note that the higher access to what-if scenarios was beneficial, and that any analysis of future specifications is likely to benefit more from our testbed implementation than it would have from successful attacks on original implementations. Additionally, we did have the TinySeRSync code available when developing and testing, and can at least attest that we did not find anything in it that suggested different behaviour than what we ultimately tested.

3.2 Testbed Implementation

We limit ourselves to an overview of the most important features of our implementation. More detail can be found in [6] and [7]. The implementation was created using the C++11 programming language. The *Boost* and *Boost.Asio* libraries were used to create efficient, easily readable and extendible code.

Participants The three participants are modeled as standalone programs interacting with each other. The first is the time *master*. On request, it provides all parameters needed for TESLA (time schedule and the key chain commitment). After initialization, it starts up two periodic timers; one sends out time packets, the other sends out keys according to the disclosure schedule. The second participant is a time *slave* that attempts to synchronize its clock. It is initialized by a request-response scheme to receive the necessary parameters. In the second step, the *loose synchronization* required by the TESLA protocol is established by performing multiple rounds of two-way time exchanges, which also quantify $\delta_{\max}$. Afterwards, the slave goes to a constant listening mode, waiting for incoming packets. From then on, the clock is only adjusted through time messages received via broadcast. The third participant, who serves as the network simulator, is a Man-In-The-Middle (MITM) attacker. It is capable of all behavior necessary to carry out the attack, such as withholding and forging packets.

The Clock We decided to run different participants on physically separate systems. This implies that different participants also have completely separate system clocks. One downside to this is the lack of a precise way of measuring the exact time difference between two clocks on separate devices. However, in our evaluation, the benefits outweighed the shortcomings: in this way, we were able to model real-world synchronization conditions as closely as possible and eliminate the concern of side effects influencing synchronization.

Clock Adjustment Algorithms We considered two different ways to adjust a clock. The first way was simply to increase the current clock value by a measured offset. The alternative was to slow down or speed up the clock for a gradual adjustment. In the context of the TESLA protocol (particularly the aspect of delayed authentication), we needed to consider a few pitfalls with respect to performing gradual adjustments. The effect of an adjustment is not yet reflected when the next packet arrives, but rather for packets a few intervals later (after the disclosure lag). Because of the disclosure lag, there is a time lag between the detection and the correction of a measured offset. Offsets will therefore be measured repeatedly. Additionally, there is the question of exactly how much the clock should be sped up or slowed down by to compensate for a detected offset. Using simple constant values can have undesirable effects, namely overcompensation (due to the correction delay mentioned above) or unnecessarily long compensation times. Our solution to this problem was to speed up or slow down the clock in such a way that it was expected to be adjusted to the given offset after $d + 1$ intervals.

3.3 Distributed Setup

For the tests, we deployed the master, slave, and MITM on two separate servers with a large distance between them. The master and the MITM were hosted on a server in the Amazon Web Services Elastic Compute Cloud in Oregon, USA. The slave was hosted on a cloud server run by the local technical university.

4 Experiment Results and Evaluation

Conforming to TinySeRSync and ASTS to the greatest extent possible, we evaluated protocol configurations regarding their susceptibility to the described attack. First, we analyzed in general which parts of the TESLA protocol are generally most relevant to the attack and formulated the core questions for our evaluation. The attack does not target the authenticity of the key packets, nor the authenticity of the time synchronization – at least not directly. Thus, we notice first that the most critical operation, and the core of the entire TESLA protocol, is the timeliness validation. Moving forward, we need to take a closer look at how the packet's timeliness is validated and what the consequences are for the security of the protocol, by the following core questions:

1. To what extent does the clock adjustment influence the timeliness validation?[3]
2. How much of a delay can be introduced so that a packet will still be timely?
3. How fast does the clock adjust, or when is the adjustment process complete?

4.1 Vulnerability of Existing Protocols to the Attack

We examined TinySeRSync and ASTS with the questions above in mind to determine whether they were vulnerable, and if so, how long it would take for an attack to succeed. Both protocols make use of a variant of TESLA. To be more precise, they make use of the µTESLA protocol, a TESLA variation specifically designed to be lightweight enough to work in wireless sensor networks. The difference between traditional TESLA and µTESLA pertains only to the way the initial parameters are distributed and secured: traditional TESLA distributes them via broadcast and secures them with asymmetric signatures, while µTESLA distributes them individually via unicast, secured by symmetric cryptography using pre-shared keys. Since none of this has any impact on the part of TESLA we wish to examine, we forego the distinction between the traditional protocol and its lightweight variant in the following analysis.

TinySeRSync The TinySeRSync protocol [15] was designed for time synchronization in wireless sensor networks and addressed the problems of security for such protocols by employing (µ)TESLA. The first step in the protocol is a secure single-hop pairwise time synchronization technique, which in this publication is called *phase I*. The second step is the actual global secure and resilient synchronization, which employs the TESLA broadcast authentication mechanism to establish a globally synchronized time throughout the entire network. This is called *phase II* in this publication. The two phases run periodically and asynchronously. The only restriction is that phase I has to be completed by a node at

[3] Since we have occasionally run into misunderstandings about this point, we would like to point out explicitly that none of our results in any way concerns the security of TESLA in a vacuum, nor that of any protocol that uses a TESLA-like mechanism to protect a generic data stream. The results apply only to protocols which use a TESLA-like mechanism to protect exactly a one-way time synchronization protocol.

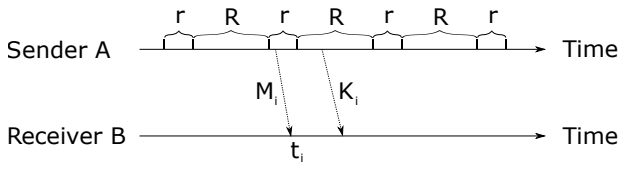


Fig. 4. Illustration of long (R) and short (r) intervals in timeliness validation context

least once before it can participate in phase II. The second phase is controlled by a source node (usually a base station), which acts as the real-time reference (master) clock in TESLA environments. The source node broadcasts the reference time periodically, with each time packet being authenticated with the TESLA protocol mechanism by the sensor nodes (slaves). Messages are re-broadcast to reach nodes that cannot directly contact the base station.

A common problem inherent to the TESLA protocol is that its use requires that loose synchronization of the clocks be established. The authors have conveniently solved this problem with the pair-wise synchronization completed in phase I. Each node knows the clock offset to each of its neighbor nodes and the maximum transmission delay for a packet, enabling timeliness validation. Even though TinySeRSync states that it utilizes (µ)TESLA, it makes some significant changes to the protocol ideas. For example, it changes the concept of the intervals and disclosure delay. Broadcast messages are still signed at one point in time and authenticated later when the key is disclosed, but the context differs slightly. Instead of using equally sized, numbered intervals, TinySeRSync uses long intervals R and short intervals r (compare Fig. 4). To be able to check the timeliness, time synchronization messages are sent exclusively in the short intervals, while key messages are sent only in long intervals. A message passes the security condition if the slave is certain that, at the receive time of the message, the key has not yet been disclosed. To ensure this, the slave simply estimates whether the sending time is within a short interval. It is important to note that, even when receiving a message carrying the global time broadcast, the timeliness is checked against the pairwise synchronized clocks. A global time broadcast is therefore merely a payload propagated throughout the network.

In order to answer Analysis Question 1 (as posed in the beginning of this section), recall that nodes running TinySeRSync have two separate clock values; one describes the clock offset between each two neighbors, while the other describes the clock difference between a node and the base station. The first clock, the offset between two neighbours, is constantly updated via unicast, which is, in the context of our attack vector, immune against attacker-induced delays.

In TinySeRSync, packets are considered timely if the slave estimates that they were sent before the end of the short interval (see Fig. 4 for an illustration). The most decisive feature of TinySeRSync in our context is that the pairwise synchronized clock is used to check timeliness, whereas the global time is not needed and therefore ignored in this part of the validation. The authors describe this technique as *local use* of the (μ)TESLA protocol. Since the pairwise synchronized

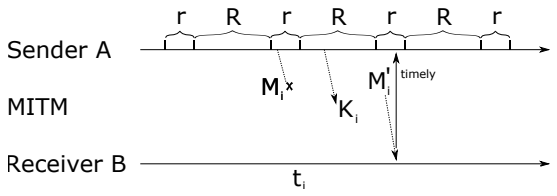


Fig. 5. TinySeRSync's simplified security condition does not differentiate between different intervals other than whether they are short or long. This can be abused.

clock is immune to attacks based on delaying packets from the base station, we expected that TinySeRSync would not be vulnerable to our attack whatsoever. Nonetheless, we set up the experiment with the master and the slave configured as described above, in order to support this expectation in practice. As expected, our attack never worked, since the delays introduced have no cumulative effect.

In conclusion, the significantly altered version of TESLA employed by TinySeRSync does not suffer from vulnerability to the attack proposed. However, it is vulnerable to a related, much simpler attack technique.

Further Security Analysis on TinySeRSync The TinySeRSync protocol has implemented the idea of reducing the maximum attack delay. It also allows a packet to arrive only in a short part of the interval. However, an oversimplification introduces a conceptional flaw and creates a vulnerability. TinySeRSync changes the notion of TESLA intervals by deploying different (short and long) intervals, but does so without any numbering of the intervals. TinySeRSync's timeliness condition is $t_i - T_0 + \Delta_{A,B} + \delta_{\max} < i(R + r) + r$, where $\Delta_{A,B}$ is the pairwise offset and $\delta_{\max}$ the maximum synchronization uncertainty. This condition ensures that a received message has been sent in a short interval, checking it against the pairwise synchronized clock. The corresponding key of the received message has to be sent and received in the following long interval.

Surveying the timeliness condition, we noticed the lack of any numeration of the intervals. Because of this, the slave can only calculate whether the packet was sent in *a* short interval, but cannot distinguish between different short intervals.

Consider the following attack, for which Fig. 5 provides an illustration. If a packet arrives in a short interval different from the one it was sent in, it will still be accepted as timely. If an attacker delays a packet in such a way that it is late by exactly $r + R$, the timeliness condition will be satisfied, even though the key for that packet has already been disclosed. The security of the protocol is therefore compromised.

We included TinySeRSync's security condition (which is referred to as *pass_secure_time_check* in the TinySeRSync code base) in our implementation and had the MITM successfully insert bogus packets. Hence, attackers with delay capabilities can control a slave's clock almost from the very start of the protocol.

ASTS The authors of the ASTS Protocol considered the TinySeRSync protocol to be unnecessarily complex, as it deploys two separate time synchronization mechanisms. Consequently, they introduced a protocol for time synchronization in wireless sensor networks, which is considerably more lightweight and stated to be more accurate [17]. Like the TinySeRSync protocol, it makes use of the (µ)TESLA protocol, but forgoes the secure pair-wise synchronization to achieve its agility. A very simple initial global broadcast is used instead to satisfy the requirement of loose synchronization and to distribute the TESLA parameters. This initial step is executed only once and secured by a single pre-shared key known throughout the network, which is valid only for this first step. Afterwards, the TESLA mechanism as described in [12] is carried out to achieve global time synchronization.

In contrast to TinySeRSync, the ASTS protocol is based on the regular (μ)TESLA protocol with no significant variations. This implies it is likely that the results on ASTS can be generalized to include more generic protocols that employ TESLA-secured one-way synchronization. Most importantly, ASTS has no extra pairwise synchronization with a separate clock mechanism. Only one global clock is adjusted with packets that are broadcasted by the base station. A broadcasted time packet is propagated throughout the network, with each node adding its processing delay to the packet and then rebroadcasting the modified packet to the neighbor nodes. A receving node calculates the offset to the base station by comparing the arrival time with the packet timestamp under consideration of the processing delay. The local clock is then adjusted by the resulting offset. Therefore, in answer to Question 1, we conclude that the regular clock adjustment process does influence the timeliness validation.

In order to answer Question 2, we examine the timeliness validation of ASTS, which is described in [17] as follows: Assume a node receives a packet at time t_{arrive} in interval i. The latest possible sending time t_{send} of this packet is $t_{\text{arrive}} + \Theta_{\max}$, where $\Theta_{\max}$ is the difference of the maximum clock difference between the two participants and the maximum network delay of the packet. The protocol suggests using a $\Theta_{\max}$ large enough to be an upper bound of any usual network delay. This value of $\Theta_{\max}$ should still be negligible compared to T_Δ, thus not influencing the following equation. The sending interval can always be calculated as $i_{\text{send}} = \lceil (t_{\text{send}} - T_0)/T_\Delta \rceil$, where T_0 is the interval starting time and T_Δ is the interval length. The timeliness condition is thus: $x < i + d$. A difference of exactly one interval exists between the timeliness of ASTS and original TESLA equations, due to the changed rounding function. This makes ASTS stricter when it comes to validating the timeliness of packets; the time a packet can be delayed by is smaller by one interval length. Given an endpoint delay δ_e, the maximum attack delay δ_{atk} that the MITM can introduce is therefore given by $\delta_{\text{atk}} = T_\Delta(d-1) - (\delta_e + \delta_{\max})$, as shown in Fig. 6. With this knowledge, we can state three dependencies:

- The maximum delay that can be added is proportional to the interval length.
- The maximum delay that can be added is proportional to the disclosure lag.

Fig. 6. Illustration of the difference between the original TESLA (right) security condition, and the slightly changed version of ASTS (left).

– There is a correlation between the maximum clock error $\delta_{\max}$ and the maximum attack delay since increasing $\delta_{\max}$ shortens the time to the next interval.

The authors of ASTS do not specify the way in which the clock is adjusted. We therefore assume, in response to Question 3, that the clock adjustment process is instantaneous, equivalent to *stepping* the clock.

Having answered all of the above questions, we deduce that ASTS is indeed vulnerable to our attack in theory. We conducted a set of experimental attack runs on our implementation in simulation of ASTS to prove this in practice and to evaluate the dependencies stated above. The overall result of the experimental attack runs confirmed that ASTS, as simulated with our implementation, is vulnerable to the attack as specified in [16]. In fact, we concluded that an intelligent attacker carrying out an attack on ASTS would eventually always be successful, regardless of ASTS' exact parameters. Nevertheless, the implications of tuning the different parameters were of great relevance and allowed us valuable insight into potential countermeasures.

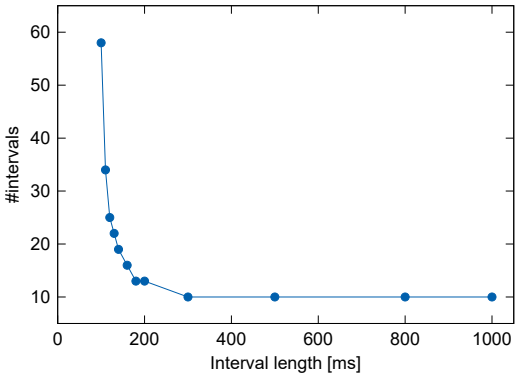
4.2 Observations from Test Attack Runs on ASTS

As mentioned earlier, ASTS employs (μ)TESLA protection of one-way time synchronization in such a straightforward way that it is suitable as an example of more generic assessments regarding the security of the scheme. Therefore, many conclusions can be generalized from the data of our attack runs on ASTS, which may thus have implications for ongoing or future specification work.

The results of carrying out the experiments with a disclosure delay of $d = 2$ and with $d = 3$ are shown in Fig. 7. In the tables, T_Δ denotes the length of the intervals, δ_{atk} denotes the maximum amount of delay the attacker can introduce with each increment, I_{suc} denotes the number of the earliest interval in which the attack was fully successful, and I_{ph2} denotes the number of the earliest interval after which Phase One of the attack was completed. Note that each of the rows in the tables represents large numbers of successful attacks, from which I_{suc} and I_{ph2} were deduced empirically.

Observations Regarding Duration of the Attack We have made a few observations regarding the duration of successful attack runs and their implications.

#	T_Δ	δ_{atk}	I_{suc}	I_{ph2}
1	1000 ms	911 ms	10	7
2	800 ms	711 ms	10	7
3	500 ms	411 ms	10	7
4	300 ms	211 ms	10	7
5	200 ms	111 ms	13	10
6	180 ms	91 ms	13	10
7	160 ms	71 ms	16	13
8	140 ms	51 ms	19	16
9	130 ms	41 ms	22	19
10	120 ms	31 ms	25	22
11	110 ms	21 ms	34	31
12	100 ms	11 ms	58	55



#	T_Δ	δ_{atk}	I_{suc}	I_{ph2}
1	1000 ms	1911 ms	14	9
2	800 ms	1511 ms	14	9
3	500 ms	911 ms	14	9
4	300 ms	511 ms	14	9
5	200 ms	311 ms	14	9
6	180 ms	271 ms	14	9
7	160 ms	231 ms	14	9
8	140 ms	191 ms	14	9
9	130 ms	171 ms	18	13
10	120 ms	151 ms	18	13
11	110 ms	131 ms	18	13
12	100 ms	111 ms	18	13

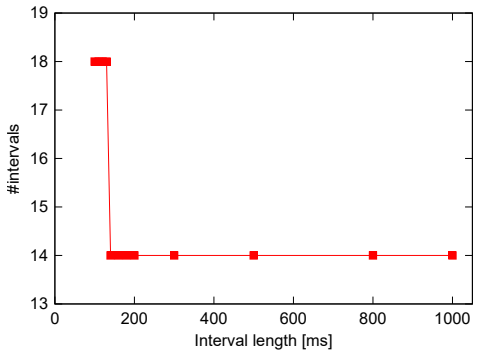


Fig. 7. Data and graphs of attack runs on ASTS with $d = 2$ (top) and $d = 3$ (bottom). Here, T_Δ denotes interval length, δ_{atk} denotes maximum introducable delay per increment, I_{suc} denotes the interval number of the earliest successful attack, and I_{ph2} denotes the interval number of the earliest successful Phase One.

These observations have been very helpful in reasoning about countermeasures (see below).

Number of Intervals for Success has Lower Bound Our first observation was that, given a disclosure delay d, there seemed to be a fixed minimum number of intervals required for the attack to succeed. Investigating further, we have found this to be caused by a combination of the protocol environment and self-inflicted limits of the attacker due to its behavioral model: it takes the next step only when it is sure (from conservative calculations) that the previous step was successful. We also refined an equation to calculate the minimum amount of intervals necessary to successfully carry out the attack (see below).

More than Just Desynchronization Our second observation is that the desynchronization condition alone is not sufficient. There also needs to be a sufficient

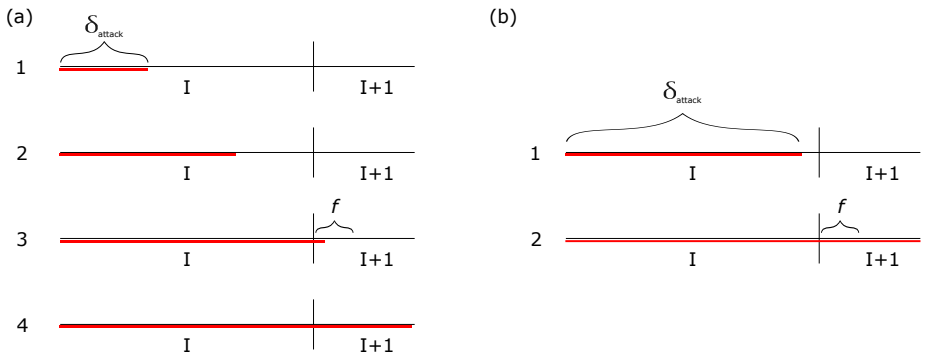


Fig. 8. Visualization of introducing delays during the attack, where f is the amount of time required to forge a bogus packet from intercepted data. In part (a), the delay introduced is small relative to T_Δ; in (b), it is relatively large.

amount of time f for the bogus packet to be considered timely. In [16], the need for f was already mentioned, but not quantified. We supply this quantification by demanding that f satisfy the following condition: $f > \delta_{\max} + \delta_e$.

The Role of Endpoint Delay and Uncertainty The third observation is that the smaller the difference $(\delta_{\max} + \delta_e) - T_\Delta$, the more iterations of step 3 are needed. In Fig. 8, we illustrate how the attacker comes closer to the goal with each iteration. If it tried to deliver a bogus packet after three iterations, it would be discarded as being untimely, even though the desynchronization condition is satisfied. Thus, another iteration is needed.

Quantification of Minimum Required Intervals The experiments with attack runs on ASTS have enabled us to deduce an equation for the earliest interval I_{suc} by which the attack's first phase can have succeeded:

$$I_{\text{suc}} = \left\lceil \frac{T_\Delta(d-1) + \delta_{\max} + \delta_e}{\delta_{\text{atk}}} \right\rceil (d+1) + 2d. \tag{1}$$

This represents a refinement over the quantification in [16] due to the ability of our experiments to account for real-world parameters such as $\delta_{\max}$, δ_e, and f.

4.3 Countermeasures and Best Practice

With a few changes to the security condition and the protocol flow, we can successfully mitigate the attack described above. Note that the changes required are significant: complete mitigation comes at the expense of introducing an extra step in the protocol, and this step must involve two-way communication. However, the contermeasures were designed to not negatively affect the quality of the achieved time synchronization, and our experiments have supported our belief that this goal was fulfilled.

Individual Measures There are two overall building blocks that we examined for their capability to prevent the attack. It turns out that it is only in combination of the two that the attack is fully defended against.

Make the Attack Take Longer The first general measure is to increase the amount of time required for successful execution of the attack. This can be achieved by combining the following tactics:

- The maximum introducable delay δ_{atk}, should be minimized. For example, one can simply set a limit to the amount of measured offset that is accepted.
- The interval length should be maximized, under the conditions that the synchronization should still work, and that the maximum δ_{atk} should not be increased. One option is to introduce dead space into the intervals where no communication can take place. This would not make sense for most application areas of the TESLA protocol, but does make sense for time synchronization.

Note that such measures alone can only delay the success of Phase One of the attack, not fully prevent it. This brings us to the second general measure.

Introduce a Periodical Synchronization Reset The second general measure is to introduce a reset to the time synchronization. The protocol must resynchronize via an alternate technique before the time by which the attack can have succeeded. This could be tied to the end of the key chain, for the sake of convenience. Note that, to provide sufficient guarantees, a reset must entail synchronization via alternative (two-way) communication. Only then can an upper bound be determined for the possible synchronization error. Note that doing this without a good understanding of how long the attack takes to be successful (or with too low an amount in that regard), this measure is wasteful. Any system that largely relies on broadcast probably has good reasons for doing so, and requiring additional two-way communication too often likely defeats the purpose.

Combined Approach To achieve the goal of completely preventing the attack, the measures described above work only in combination. The first measure gives a guarantee that the protocol is secure up to a given point in time. The second measure ensures that the protocol (including the security guarantee above) is reset before that point in time is reached. From another perspective, the second measure makes the attacker start over with its scheme. The first measure ensures that there is a reasonable point in time to do so. This can neither be too late (which would leave security compromised) nor too early (which would be wasteful of communication resources).

Our experiments have suggested that if one of the measures is not taken, or not taken correctly, the attack can always succeed eventually. The converse is also true: in our experiments, if both measures are taken correctly, the attack can never be executed successfully.

4.4 Recommendations for Fixing TinySeRSync and ASTS

From the above, we deduce that in order to fix ASTS' security, one would have to re-introduce a form of pairwise synchronization. Note that this might render stated advantages of ASTS over TinySeRSync invalid.

For TinySeRSync, it seems that the first fix would be to re-introduce the actual numbering of intervals that (μ)TESLA originally prescribes. Given this modification, it seems to us that there would be a reasonable way to combine the two protocol phases (one of which employs two-way time synchronization anyway) to a successful variant of TESLA-protected synchronization.

5 Conclusions

5.1 Summary

We have investigated security issues with the deployment of the TESLA protocol in one-way time synchronization protocols. To this end, we have built an implementation that reflects the security-relevant characteristics of the class of protocols that apply delayed disclosure authentication (e. g. TESLA) to one-way time synchronization. This has enabled us to draw a number of conclusions (presented in Section 4), some of which are specific to certain protocols, while others hold generally for the whole class of protocols.

We have conducted a range of tests proving that the attack described in [16] can be executed in a real-world environment with physically distributed devices on a protocol that represents a faithful recreation of ASTS. Furthermore, we have conducted tests proving that TinySeRSync's security measures can be broken, albeit by an attack different from our original concept. Regarding the more general class of protocols, we have supported the theory that, without extra security features beyond those that TESLA provides, the attack can always succeed eventually. In particular, we have reaffirmed our conjecture that the attack cannot be prevented with only one-way communication.

We have also collected a set of countermeasures, some of which can be used to reduce the amount of delay the attacker can introduce, making the attack take longer. In Equation 1, we quantify the minimum amount of time/intervals the attack will take to execute until packets can be forged at will. With this amount known beforehand, additional countermeasures can be taken early enough that the attack can be completely defeated. However, this still comes with the strict caveat that these additional measures need to involve two-way communication. All of these implementation-based results represent improvements on the work in [16], where the attack was only investigated via model checking.

5.2 Future Work

All relevant in-development or future time synchronization protocol specifications should be analyzed regarding their vulnerability to the attack. It is our hope that the implementation created for this work can support such analyses.

Additionally, further research regarding potential countermeasures would be interesting. Even though we are skeptical of the existence of efficient countermeasures that do not require two-way communication, the value of such a design would be tremendous, especially in environments such as global navigation satellite systems, where communication is designed to be one-way only.

Overall, it should be stated that even with the highlighted issues, we still believe that TESLA is a worthwhile candidate for a protection mechanism for one-way synchronization. Therefore, the next research project we are focusing on are formal proofs of positive statements about the security of TESLA-protected one-way synchronization protocols. Such statements might include that, other than the attack vector described in [16], TESLA-protection of one-way synchronization protocols is secure and/or that TESLA-protection with a given set of added countermeasures overall makes one-way synchronization secure.

Acknowledgment

We would like to express our thanks to Dieter Sibold, for supervising large parts of the work and for valuable input in the creation of this paper.

References

1. Standard for a precision clock synchronization protocol for networked measurement and control systems, `https://standards.ieee.org/develop/project/1588.html`
2. Annessi, R., Fabini, J., Zseby, T.: It's about time: Securing broadcast time synchronization with data origin authentication. In: 2017 26th International Conference on Computer Communication and Networks (ICCCN). pp. 1–11 (July 2017). https://doi.org/10.1109/ICCCN.2017.8038418
3. Annessi, R., Fabini, J., Zseby, T.: SecureTime: Secure multicast time synchronization. ArXiv e-prints (May 2017)
4. Dolev, D., Yao, A.: On the security of public key protocols. IEEE Transactions on Information Theory **29**(2), 198–208 (1983)
5. Fernandez-Hernandez, I., Rijmen, V., Seco-Granados, G., Sim'on, J., Rodríguez, I., David Calle, J.: A navigation message authentication proposal for the galileo open service. Navigation - Journal of The Institute of Navigation **63** (03 2016)
6. Hildermeier, G.: Attacking tesla-secured time synchronisation protocols. Master's Thesis (09 2017)
7. Hildermeier, G.: Testbed implementation for simunlating attacks on tesla-secured time synchronisation protocols (09 2017), `https://gitlab1.ptb.de/teiche04/Hildermeier-TESLA-Protected-One-Way-Synchronization.git`
8. Levine, J.: A review of time and frequency transfer methods. Metrologia **45**(6), S162–S174 (2008). https://doi.org/10.1088/0026-1394/45/6/S22, `<GotoISI>://WOS:000262502900023http://iopscience.iop.org/0026-1394/45/6/S22/pdf/0026-1394_45_6_S22.pdf`
9. Mills, D.L.: Internet time synchronization: the network time protocol. IEEE Transactions on Communications **39**(10), 1482–1493 (1991)
10. Perrig, A., Canetti, R., Tygar, J.D., Song, D.: Efficient authentication and signing of multicast streams over lossy channels. In: Security and Privacy, 2000. S&P 2000. Proceedings. 2000 IEEE Symposium on. pp. 56–73. IEEE (2000)

11. Perrig, A., Canetti, R., Tygar, J.D., Song, D.: The TESLA broadcast authentication protocol. RSA Cryptobytes **5** (2005)
12. Perrig, A., Szewczyk, R., Tygar, J.D., Wen, V., Culler, D.E.: SPINS: Security protocols for sensor networks. Wireless Networks **8**(5), 521–534 (2002)
13. Sibold, D., Roettger, S., Teichel, K.: Network Time Security. Internet Draft draft-ietf-ntp-network-time-security-15, Internet Engineering Task Force (Sep 2016), `https://datatracker.ietf.org/doc/html/draft-ietf-ntp-network-time-security-15`, work in Progress
14. Sibold, D., Roettger, S., Teichel, K.: Using the Network Time Security Specification to Secure the Network Time Protocol. Internet Draft draft-ietf-ntp-using-nts-for-ntp-06, Internet Engineering Task Force (Sep 2016), `https://datatracker.ietf.org/doc/html/draft-ietf-ntp-using-nts-for-ntp-06`, work in Progress
15. Sun, K., Ning, P., Wang, C.: TinySeRSync: Secure and resilient time synchronization in wireless sensor networks. In: Proceedings of the 13th ACM Conference on Computer and Communications Security. pp. 264–277. ACM (2006)
16. Teichel, K., Sibold, D., Milius, S.: An attack possibility on time synchronization protocols secured with TESLA-like mechanisms. In: Information Systems Security, pp. 3–22. Springer (2016)
17. Yin, X., Qi, W., Fu, F.: ASTS: An agile secure time synchronization protocol for wireless sensor networks. In: Wireless Communications, Networking and Mobile Computing, 2007. WiCom 2007. International Conference on. pp. 2808–2811. IEEE (2007)