Delayed Authentication and Delayed Measurement Application in One-Way Synchronization

K. Teichel, D. Sibold, 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.

Full Citation of the original article published by IEEE:

K. Teichel, D. Sibold and G. Hildermeier, "Delayed Authentication and Delayed Measurement Application in One-Way Synchronization," *2018 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control, and Communication (ISPCS)*, Geneva, 2018, pp. 1-6.
doi: 10.1109/ISPCS.2018.8543083

Available at:
https://doi.org/10.1109/ISPCS.2018.8543083

Delayed Authentication and Delayed Measurement Application in One-Way Synchronization

Kristof Teichel and Dieter Sibold
Physikalisch-Technische Bundesanstalt
Braunschweig, Germany
Email: kristof.teichel@ptb.de / dieter.sibold@ptb.de

Gregor Hildermeier
Technische Universität Braunschweig
Braunschweig, Germany

***Abstract*—We have performed an analysis of an attack vector regarding one-way time synchronization protocols protected by TESLA-like mechanisms. In this paper, we present our testbed implementation that allows simulation of such protocols to take place, and we give an overview of the results regarding vulnerability of existing protocols, potential countermeasures, and relevance to specifications currently in development. We omit much of the cryptography-related details of our security analysis in favor of specifics regarding the effects and pitfalls related to the delayed authentication used by all TESLA-like protection mechanisms. We place particular emphasis on implications concerning the security and time performance of immediate versus delayed measurement application where this occured in our evaluation.**

I. Introduction

With the importance of cybersecurity increasing in recent years, demand for authenticity in time synchronization protocols has grown. However, providing authenticity in a time synchronization context has proven to be difficult. On the one hand, the number of slaves can be vast, which rules out authenticity measures using conventional symmetric-key cryptography. On the other hand, asymmetric cryptography cannot always be applied, especially on devices with low computational power or via connections with limited bandwidth. To meet this challenge, a mechanism that uses simple symmetric cryptography to authenticate data in large networks has been adopted for time synchronization. The mechanism is called Timed Efficient Stream Loss-tolerant Authentication (TESLA) [1], [2] and makes use of delayed authentication.

At least two published protocols already make use of the TESLA mechanism to secure time-related messages: the "Secure and Resilient Time Synchronization protocol" (TinySeRSync) [3] and the "Agile Secure Time Synchronization protocol" (ASTS) [4], both of which are meant for wireless sensor networks. Furthermore, the time community is reviewing the application of TESLA for ongoing specification work (see Section V for a list of relevant specifications).

However, a potential attack vector has been discovered where, under certain circumstances, the authenticity provided by TESLA used for time synchronization via broadcast can be compromised entirely [5]. We carried out an evaluation of this attack with the help of an independently developed testbed implementation that let us simulate the behavior of TESLA-protected one-way time synchronization protocols and the behavior of attackers working to break those protocols. With our implementation, we tested dependencies between protocol parameters to find out in which scenarios the attack is or is not successful. Furthermore, we deduced a set of countermeasures to mitigate the attack; these contermeasures can be included in future specifications of time synchronization protocols involving one-way (e.g. multicast) communication.

In this paper, we present a brief overview of the results of our security analysis: we report on vulnerabilities of the existing protocols we were able to evaluate with our simulations. Furthermore, we list protocols currently in development for which our analysis might become relevant because they are likely to employ TESLA-like mechanisms. The cryptography-related details of our security analysis can be found in a separate publication [6] and in the master's thesis that encompassed this work [7]. We therefore omit them here in favor of our findings on immediate versus delayed measurement application. Having only delayed authentication available (an essential characteristic of TESLA-like mechanisms) raises the delicate issue of how to apply the time measurements obtained – immediately, when they are still unauthenticated, or after the potentially long delay that it takes to confirm their authenticity? This question is interesting in two different respects: it affects how fast the attack can be successful, and it influences the performance of the secured time synchronization protocol.

This paper is structured as follows: Section II provides an introduction to the TESLA protocol as well as the attack vector that we base our simulation and analysis on. The testbed implementation that we employed is introduced and outlined in Section III. In Section IV, we present the results of our evaluation. This comprises the overview of our analysis regarding protocol vulnerability and countermeasures, as well as our findings related to immediate versus delayed measurement application. Section V presents relevant ongoing specification work, and Section VI concludes the paper.

II. Preliminaries

This section provides an introduction to how the TESLA protocol works, as well as a schematic description of how the central attack vector is constructed.

A. TESLA Overview

The TESLA protocol [1] was originally designed to authenticate media streams. It was created to mitigate the typical pro-

blems entailed by symmetric and asymmetric cryptography: key distribution (regarding the former) and considerable required computing power (regarding the latter). Using TESLA, the slave can authenticate any packet sent by the master, as long as it arrives within a certain timeframe. Time plays an essential role, as explained in more detail below.

1) Basic Concept: The TESLA protocol 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 disclosed at a later point in time, after a time d, the so-called *disclosure lag*. A received packet is buffered and, as soon as the key is disclosed, it can be authenticated. As long as the slave is certain that the key was not disclosed before the packet was received (a condition called the *timeliness* of the packet), the authenticity of the signature is guaranteed. Using this procedure, symmetric-key cryptography can be used in a broadcast setting, thus avoiding the problems of asymmetric cryptography. To achieve this, the TESLA protocol requires an initial time synchronization in which a *rough upper bound* $\delta_{\max}$ of the clock offset is determined. In order for TESLA to work, two main concepts are deployed:

a) Commitment with a Key Chain: The concept of the key chain makes it possible to commit to a key before using it, and also enables packet loss toleration. 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 only by its predecessors. The master applies F', another pseudo-random function, to a *key value* K_i 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}, F, and F' securely.

b) 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 will be associated with the key value K_i. Every packet sent in interval I_i is signed with the same interval key. In interval I_{i+d}, the key of I_i is then disclosed. At any point in time t, a network participant can compute the interval index i from its clock value. A depiction of the concept of intervals is presented in Fig. 1. In the example shown, a packet is authenticated four intervals later. The packet P_j is signed with the key K'_i, since it was sent in the interval I_i. The key for I_i is disclosed in I_{i+4}.

B. Attacking TESLA-secured Time Synchronisation Protocols

The attack described in this section was first formulated in [5]. It forms the basis for the behavior of the adversary in our implementation. The attack is possible when the TESLA protocol is used to authenticate time 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.

1) Attacker Model: A powerful attacker is assumed to be an invader that operates under the Dolev-Yao attack model [8]. It is therefore capable of impersonating any member of the network as long as there is no extra authentication mechanism in place. Additionally, it is assumed to be capable of delaying any packet. This has a special significance in the context of time synchronization protocols and also in the attack that is explained below. However, it is not able to break cryptographic ciphers or reverse one-way hash functions; furthermore, nonces and secret keys cannot be guessed.

2) Attack Synopsis: The attack has two phases; *Phase One* has multiple steps:

- The attacker consistently delays each packet broadcasted by the master by d_1.
- 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 d_1 will start to take effect.
- This desynchronization increases the time by which the slave will accept a delayed packet as timely, thus enabling the attacker to delay packets by $d_1 + d_2$.
- The attacker continues to increase the delay as according to the step above.

Eventually, the clocks will desynchronize by more than $(d - 1)T_\Delta$, where d signifies the disclosure lag. Phase One of the attack is then complete.

After the end of Phase One, the adversary can perform *Phase Two* of the attack. It is then 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 adversary can fully impersonate the master; thus, the security gain of the TESLA protocol is negated.

3) Example Attack: To further illustrate the attack, it is demonstrated again below with specific values $d = 2$ and $T_\Delta = 6\,\mathrm{s}$. Note that the real-world values for TESLA, especially lengths of intervals, can vary greatly depending on the protected protocol and the frequency with which it sends messages. The master and the slave perform the secured initial exchange of information that includes the values mentioned above and the initial key chain commit K_{-1}. The master then starts to periodically broadcast the value of its clock at the beginning of each interval I_i at the time s_i.

a) Phase One: This attack phase is depicted in Fig. 2.

- The attacker delays P_0 by $d_1 = 4\,\mathrm{s}$. The slave checks the timeliness condition and accepts P_0 as timely.
- Packets P_1 and P_2 are delayed in the same way as P_0. When the slave receives P_2, it extracts the included key value K_0. The slave (successfully) authenticates P_0, concludes that its clock is fast by four seconds, and adjusts it accordingly.
- The attacker can now delay the subsequent packets, starting with P_3, by $d_1 + d_2 = 8\,\mathrm{s}$.
- In our example, the increase in step 3 only has to be executed once, since $8\,\mathrm{s} > T_\Delta$.

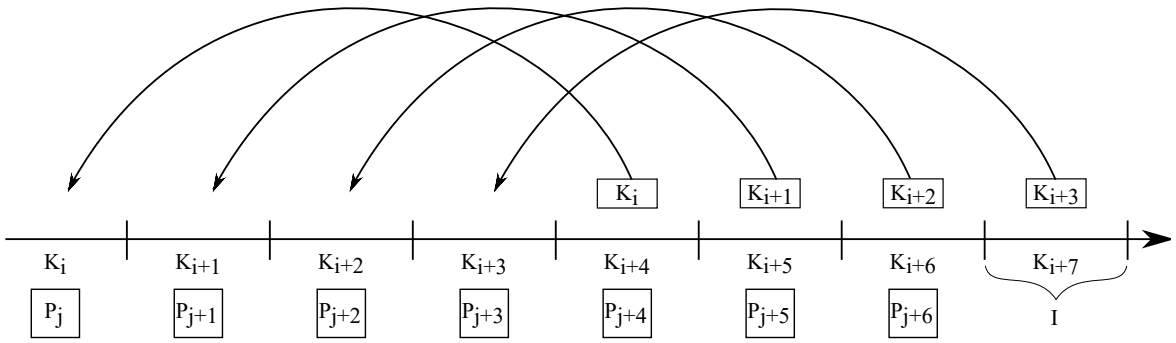

Fig. 1. Depiction of how TESLA authenticates packets with keys carried by subsequent packets.

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 is sufficient and Phase One is therefore complete.

b) Phase Two: can now begin, since there is an offset of more than one interval between the two clocks. The attacker blocks P_6 and the following packets from ever reaching the slave. He waits until the master discloses K_6 in interval I_8 (which is sent at $t = 48$ s), extracts K_6 and derives K_6'. It can now forge a packet Q_6 with any chosen bogus time, use K_6' to create a valid MAC and forward it to the slave. In this case, it has a time window of $T_\Delta/3$ to deliver the packet. In another forged packet Q_8 he includes the original and correct key value K_6. The slave will use this key to (successfully) authenticate the bogus packet Q_6, since the key value is a valid link in the key chain. The slave now uses the bogus time data in Q_6 to adjust its clock, and the attack is therefore successful.

III. Our Testbed Implementation

We have created a new implementation of a highly configurable TESLA-protected one-way synchronization protocol. Its main purpose is to simulate the behavior of specific specimen of such protocols. An overview of the implementation is given here; more detail can be found in [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.

A. Model of the Participants

There are three standalone programs that interact with each other. The first is the time *master*. On request, it provides all parameters needed for TESLA, such as the schedule and the key chain commit. After initialization, it starts two periodic timers; one sends out synchronization packets, the other sends out keys according to the disclosure lag.

The second is a time *receiver*, also called a *slave*, which attempts to synchronize its clock. It is initialized by using a request-response scheme to receive the necessary parameters. In the second step, the *loose synchronization* required by the TESLA protocol is established via multiple rounds of two-way time exchanges, which also measure $\delta_{\max}$. After the setup, the slave changes to a constant listening mode, waiting for incoming packets. From then on, the clock is only adjusted through time synchronization messages received via broadcast.

Below, we introduce a third participant who serves as the network simulator and also as a powerful "man in the middle". It is capable of all the behavior necessary to carry out the attack, such as withholding packets and forging packets that fit seamlessly into the original packet stream.

B. The Clock

Instead of simulating a clock on the user level, we decided to use two actual, separate system clocks in the implementation in order to model real-world synchronization conditions as closely as possible. The downside to this is the lack of a precise way of measuring the exact time difference between the two clocks.

We considered two different ways to adjust a clock. The first is simply to increase the current clock value by a measured offset. The alternative is 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 several shortcomings associated with performing gradual adjustments. The effect of an adjustment is not yet reflected when the next packet arrives, but when packets arrive 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 was to speed up or slow down the clock to such an extent that it is expected to be adjusted to the given offset after $d+1$ intervals.

IV. Evaluation Results

In this section, we present the results we obtained with our testbed implementation. We include brief overviews of

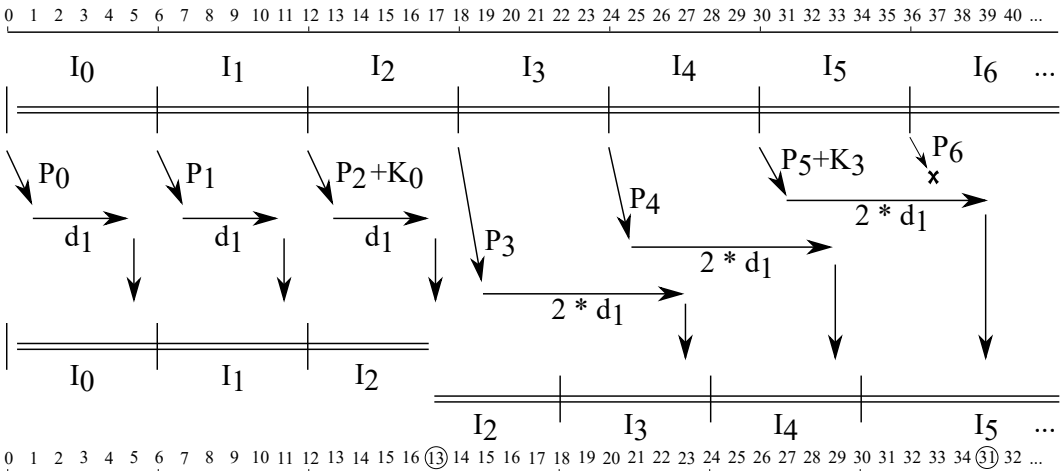

Fig. 2. Illustration of the attack on an example network.

our results on protocol vulnerability, possible countermeasures and potential future time synchronization specifications that might be affected. We then present our findings on delayed versus immediate application of received packets, as well as a proposal for how to combine the two techniques.

A. Vulnerabilty of Existing Protocols

Our analysis concludes that both of the examined protocols are vulnerable. For ASTS, the attack can be executed very similarly to what is described in [5]. No matter what parameters are chosen, the attacker is able to succeed eventually, even if it might take a long time. With TinySeRSync, the situation is more intricate. We were unable to approach this situation with the scheme from [5], instead learning about techniques that could provide a valid defense. However, the robustness to the attack was due to changes to the TESLA scheme that make the protocol vulnerable to other, much simpler attack techniques that also involved packet delay tactics. More detail on our security analysis can be found in [6] and [7].

B. Countermeasures and Best Practice

With a few changes to the timeliness condition and the protocol flow, we were able to successfully mitigate the attack described above. However, the changes required are significant: complete mitigation comes at the expense of there being an extra step in the protocol, and the fact that this step must involve two-way communication. More details on the countermeasures presented can be found in [7].

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

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

where δ_{atk} is the maximum amount of additional delay that the attacker can insert per interval, and δ_e is the natural endpoint delay between the given slave and master. This represents a refinement over the quantification in [5] due to the ability of our experiments to account for real-world parameters.

2) Make the Attack Take Longer: The first general measure is to increase the time required for execution of the attack. In general, this can be achieved by making it harder, e. g. via time source diversity. This can be achieved by combining the following tactics:

- The maximum delay δ_{atk} that can be introduced 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 time into the intervals where no communication can take place. This would not make sense for most application areas of the TESLA protocol, but it does make sense for time synchronization.
- The timeliness condition should be independent from d.

3) Introduce a Periodical Reset to Time Synchronization: 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. Note that, to be effective, a reset must entail synchronization via alternative (two-way) communication. A reset as described here can and should still be realized via gradual clock adjustment.

4) 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. Our experiments have suggested that, if one of the measures is not taken, or is 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 will never be executed successfully.

C. Clock Adjustment Algorithm

In this Section, we test strategies for clock adjustment to examine the effects and interdepndencies of gradual clock adjustment together with a disclosure delay. Note that we always use gradual adjustment over stepping the clock. The two approaches differ in when measurements are used to start such a gradual adjustment.

1) Dynamic Delayed Adjustment: The "SlewClock Algorithm" was designed to find a value of how much the clock should be sped up or slowed down by to allow offsets of any size to be corrected within a reasonable time (i. e., the clock should be adjusted by a measured offset after $d+1$ intervals). The algorithm calculates this value dynamically, depending on the given offset δ_{meas}, to adjust it to the interval length T_Δ and the disclosure delay d. The adjustment required per second t_{adj} is calculated as follows:

$$t_{\text{adj}} = \frac{\delta_{\text{meas}}}{(d+1)T_\Delta}. \tag{2}$$

The way in which the adjustment by t_{adj} is further realized depends on the setup of the device and the operating system.

We set up two sets of experiments, both with an interval length of one second and a disclosure lag of 2. Time synchronization packets are sent once per interval. The network introduces a delay for every packet of 100 ms in one experiment, and of 500 ms in the other. The offset the slave calculates for each incoming packet is measured. The results show how the clock gradually adjusts to the offset, but then desynchronizes again, which is called the *swinging effect*.

In both experiments, the results are the same: the swing falls below 1 ms after 35 packets. The measurements of the experiments with delay 100 ms are plotted in Fig. 3 (left).

However, there is a limit in our implementation to how much the clock can be sped up or slowed down by. The underlying system call *adjtimex()* does not accept a change of more than 10 %, which is about 6 s per minute. When there is a need to adjust to very large offsets, the slew clock algorithm is thus not suitable; in these cases, the clock should be stepped (i. e. simply set to the new value instead of gradual adjustment). Ideally in a time synchronization protocol with real-world parameters, a mix of slewing and stepping should be implemented.

Regarding the attack described, a slave that uses the slew-clock algorithm to adjust to offsets is still vulnerable. The key difference is that the MITM has to be configured more precisely and more carefully, since it is harder to estimate the slave's clock. Futhermore, the attack takes considerably longer, since the MITM has to wait for the slave to adjust to an introduced offset, which can take as many as 35 intervals, as stated above.

2) Immediate Adjust: To mitigate the inaccuracy caused by the *swinging effect*, the clock could adjust immediately upon reception of the time synchronization packet. Obviously, this constitutes problematic, dangerous behavior, since any information is accepted even if it later turns out to be malicious. This approach can only work if the time synchronization packets are not encrypted with the corresponding keys but only signed instead. In our implementation, this is the case; by setting the configuration to immediate adjustment, the slave handles received time synchronization packets as soon as it gets them. The packets are still buffered for later validation; however, at the time of the validation, the packet has already been processed. We set up the same experiment described in the section above, with $T_\Delta = 1$ s, $d = 2$, and introduced an offset of 100 ms. A plot of the measurements can be found in Fig. 3 (right). The results show how the clock slowly adjusts to the offset in an ideal curve. Within 17 packets at the latest, the clock offset falls below 1 ms.

We conclude from these experiments that the "SlewClock Algorithm" works very well in order to find a value to complete the adjustment process within a reasonable time. Due to the disclosure delay, the swinging effect occurs, which roughly doubles the intervals needed to completely adjust to an offset. In all of the experiments, the offset eventually approached zero with a deviation of 400 μs.

If the approach involving immediate adjustment were to be deployed in a real-world scenario, one would need to carefully balance the pros and cons. A simple idea is to use this method to adjust to offsets until a cryptographic check fails, and at that time to switch back to the regular TESLA procedure of delayed adjustment. If a small disclosure lag is chosen, the damage caused by accepting fraudulent packets may be manageable. The advantage is obvious: if no cryptographic alarm is triggered, the time synchronization accuracy will improve, as shown above. This approach is merely an idea, and needs to undergo more testing to determine the extent to which it is applicable in real-world scenarios.

V. Relevance for Specifications in Development

TinySeRSync and ASTS are not the only time synchronization protocols to which our analysis is relevant, even though they are the only ones we are aware of whose specifications have already been completed. Overall, the adoption of TESLA-like security mechanisms in the time synchronization world has been widespread and is present in several specifications currently in development.

- The IEEE uses a TESLA-based security scheme in the upcoming version of the Precision Time Protocol (PTP) [9].
- The IETF has been discussing TESLA's use for the broadcast mode of the Network Time Protocol (NTP) [10] in the context of the ongoing Network Time Security specification [11], [12].
- The ESA is incorporating TESLA into the Open Service Message Authentication scheme [13] of Galileo, the European global navigation satellite system.

Both PTP and NTP offer innate ways to realize two-way synchronization for our countermeasures: PTP has *delay request/response* messages, NTP has *client-server* mode.

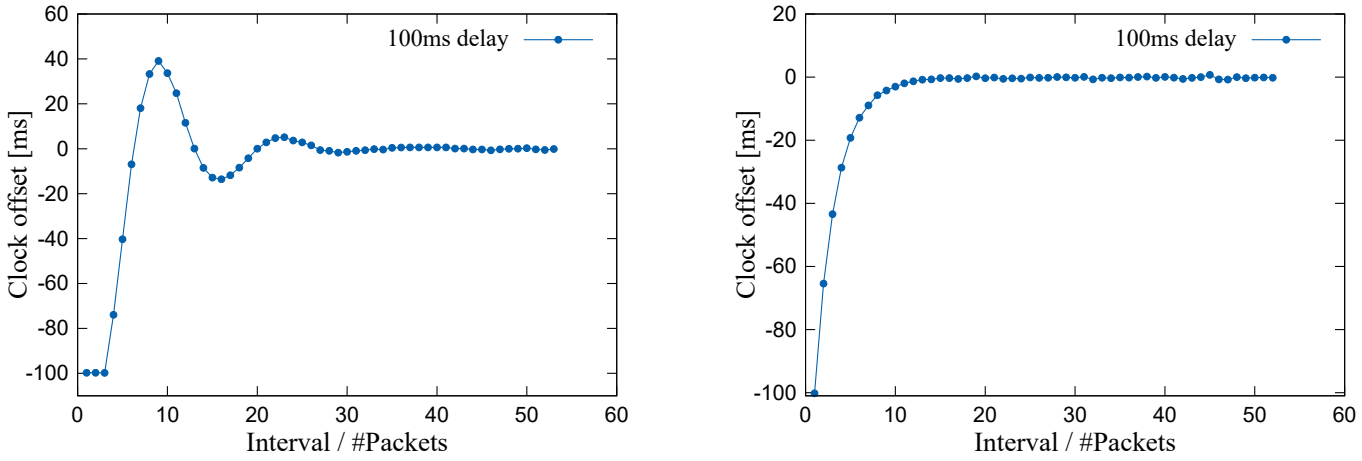

Fig. 3. Depiction of the *swinging effect* with delayed adjustment (left) versus immediate adjust (right), and an introduced offset of 100 ms. With the given setup, the number of packets on the x-axis coincides exactly with the number of intervals and with the number of seconds that have passed.

VI. Conclusion and Future Work

We have investigated problems entailed by the deployment of the TESLA protocol, one of which is the attack described in [5]. We have built an implementation of a time synchronization protocol for the Unix operating system that reflects the security-relevant characteristics of TESLA-secured one-way time synchronization protocols. Our implementation is capable of simulating different protocols regarding their security and synchronization features. We have conducted tests to prove that the theoretical attack described can also work in reality. We have found that, without extra security features, the attack is always possible. However, we have deduced that the system is secure until Phase One of the attack has succeeded, and quantified the earliest time by which this can occur. We have also stated recommendations for setting up a time synchronization protocol with TESLA-secured one-way communication.

Any other existing or future time synchronization protocol or specification using TESLA (in particular all of those listed in Section V) should be analyzed regarding the applicability of the attack presented. The implementation created for our analysis could be used for such an endeavor. Especially regarding the context of the PTP, a countermeasure relying on periodic two-way exchanges would fit very well in theory.

Additionally, we presented findings on delayed and immediate application of measured offsets for which only delayed authentication is available. It would be interesting to further pursue the idea of switching between the two behaviors dynamically according to detected threats, and to consider this idea in any new specification.

Acknowledgments

The authors would like to thank Stefan Milius for his input and ideas, as well as his help in enabling our cooperation. Special thanks are also due to David Goltzsche and Rüdiger Kapitza for their discussions and support.

References

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

[2] ——, "The tesla broadcast authentication protocol," *Rsa Cryptobytes*, vol. 5, 2005.

[3] K. Sun, P. Ning, and C. Wang, "Tinysersync: secure and resilient time synchronization in wireless sensor networks," in *Proceedings of the 13th ACM conference on Computer and communications security*. ACM, 2006, pp. 264–277.

[4] X. Yin, W. Qi, and F. Fu, "Asts: An agile secure time synchronization protocol for wireless sensor networks," in *Wireless Communications, Networking and Mobile Computing, 2007. WiCom 2007. International Conference on*. IEEE, 2007, pp. 2808–2811.

[5] K. Teichel, D. Sibold, and S. Milius, "An attack possibility on time synchronization protocols secured with tesla-like mechanisms," in *Information Systems Security*. Springer, 2016, pp. 3–22.

[6] G. Hildermeier, K. Teichel, and D. Sibold, unpublished.

[7] G. Hildermeier, "Attacking tesla-secured time synchronisation protocols," Master's Thesis, 09 2017.

[8] D. Dolev and A. Yao, "On the security of public key protocols," *IEEE Transactions on information theory*, vol. 29, no. 2, pp. 198–208, 1983.

[9] "Standard for a precision clock synchronization protocol for networked measurement and control systems," IEEE, Standard 1588. [Online]. Available: https://standards.ieee.org/develop/project/1588.html

[10] D. L. Mills, "Internet time synchronization: the network time protocol," *IEEE Transactions on communications*, vol. 39, no. 10, pp. 1482–1493, 1991.

[11] D. Sibold, S. Roettger, and K. Teichel, "Network Time Security," Internet Engineering Task Force, Internet-Draft draft-ietf-ntp-network-time-security-nn, Sep. 2016, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-ietf-ntp-network-time-security

[12] D. Franke, D. Sibold, and K. Teichel, "Network Time Security for the Network Time Protocol," Internet Engineering Task Force, Internet-Draft draft-ietf-ntp-using-nts-for-ntp, 2018, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-ietf-ntp-using-nts-for-ntp

[13] I. Fernandez-Hernandez, V. Rijmen, G. Seco-Granados, J. Simn, I. Rodrguez, and J. David Calle, "A navigation message authentication proposal for the galileo open service," *Navigation - Journal of The Institute of Navigation*, vol. 63, 03 2016.