An Attack Possibility on Time Synchronization Protocols Secured with TESLA-Like Mechanisms

Kristof Teichel[1(✉)], Dieter Sibold[1], and Stefan Milius[2]

[1] Physikalisch-Technische Bundesanstalt,
Bundesallee 100, 38116 Braunschweig, Germany
kristof.teichel@ptb.de
[2] Chair for Theoretical Computer Science,
Friedrich-Alexander Universität Erlangen-Nürnberg,
Martensstr. 3, 91058 Erlangen, Germany

Abstract. In network-based broadcast time synchronization, an important security goal is integrity protection linked with source authentication. One technique frequently used to achieve this goal is to secure the communication by means of the TESLA protocol or one of its variants. This paper presents an attack vector usable for time synchronization protocols that protect their broadcast or multicast messages in this manner. The underlying vulnerability results from interactions between timing and security that occur specifically for such protocols. We propose possible countermeasures and evaluate their respective advantages. Furthermore, we discuss our use of the UPPAAL model checker for security analysis and quantification with regard to the attack and countermeasures described, and report on the results obtained. Lastly, we review the susceptibility of three existing cryptographically protected time synchronization protocols to the attack vector discovered.

Keywords: Security protocols · Broadcast · Time synchronization protocols · TESLA · Security analysis · UPPAAL

1 Introduction

Time synchronization protocols based on broadcast or multicast play an important role in distributed computer networks such as sensor networks [24]. In many cases, protection of time synchronization packets is indispensable in order to guarantee the integrity and authenticity of time information; this is especially true in open environments like the internet. In general, the performance of time synchronization protocols decreases due to latencies caused by computational operations, in particular by cryptographic operations. This decrease in performance needs to be considered in the design of security measures (see e.g. [16]). The Timed Efficient Stream Loss-tolerant Authentication (TESLA) protocol and its variants [8,10,17–19] rely on symmetric cryptography and use delayed disclosure of keys to acquire the asymmetric properties that are desired for broadcast

I. Ray et al. (Eds.): ICISS 2016, LNCS 10063, pp. 3–22, 2016.
DOI: 10.1007/978-3-319-49806-5_1

communication. They thus fulfill the special security requirements for broadcast time synchronization and are employed by multiple time synchronization protocols. The fact that delayed disclosure requires the participants to agree on a schedule creates a challenging interaction between the security mechanisms and the time synchronization process. In this paper, we discuss the security of time synchronization broadcast associations secured via variants of the TESLA protocol, particularly with respect to this interaction. We highlight a specific attack vector that an adversary can follow to circumvent the security measures of such associations and to deliver false timing information to a participant. We give a generalized description of the attack vector and use it on a minimal example protocol for illustration. We also discuss feasible countermeasures that can either make the attack theoretically impossible (which requires a significant effort, possibly requiring a change in the communication structure) or mitigate the attack by making its execution practically impossible. Next we present a model in UPPAAL [2] that allows the analysis of the behavior of the protocol participants' clocks during an attack; this model also allows the assessment of the effectiveness of the countermeasures discussed. In addition, we investigate the extent to which specific existing time synchronization protocol specifications might be vulnerable to the attack vector discovered. One of the intentions of this paper is to facilitate discussion about the attack vector, and to supply a basis for possible further analysis.

The remainder of this paper is structured as follows: Sect. 2 provides an overview of the two main approaches to time synchronization (unicast-type and broadcast-type communication) and of the usual security measures that each approach entails. Section 3 defines the notation used throughout the paper, discusses the underlying assumptions and defines a Minimal Example Protocol (MEP) for illustration in later sections. Section 4 shows the attack vector on broadcast-type time synchronization protocols that have been secured with TESLA-like mechanisms, exemplified by means of the MEP. In Sect. 5, we discuss a selection of possible countermeasures against the attack vector shown. Section 6 presents our automated analysis in UPPAAL and provides its results; these concern, on the one hand, quantification of the parameters that allow the attack to happen and, on the other hand, the effectiveness of some of the countermeasures discussed in previous sections. Section 7 presents three examples in which TESLA-like mechanisms are employed to secure broadcast-type time synchronization: Network Time Protocol (NTP) secured by Network Time Security (NTS) [23], TinySeRSync [24], and Agile Secure Time Synchronization (ASTS) [27]. This section gives an assessment on how robust those protocols are against the attack vector under discussion. Finally, Sect. 8 concludes the paper.

2 Time Synchronization Security

2.1 Main Synchronization Techniques

There are two general methods for time synchronization [26]: using one-way time transfer and using two-way time transfer. Figure 1 depicts typical message

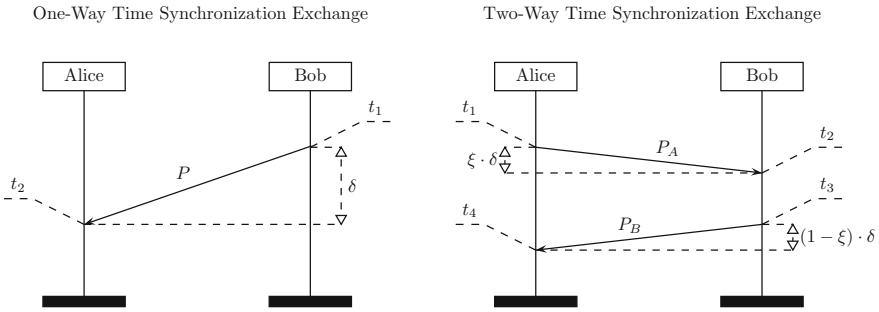

Fig. 1. Schematic depiction of typical message exchanges that are used for time synchronization between Alice and Bob. The left diagram depicts one-way synchronization, while the right diagram depicts two-way synchronization.

exchanges for these two kinds of transfer. Although there are exceptions, network protocols for time synchronization typically use two-way transfer when they employ unicast-type communication (one sender, one receiver), whereas they use one-way transfer when they employ broadcast-type communication (one sender, multiple receivers). The exchange of time synchronization protocol packets between the nodes involved is accompanied by a delivery delay δ whose characteristics depend on the underlying network. If one-way time transfer is used (Fig. 1, left diagram), messages are transmitted only from the time server to the time client. In this case, the delivery delay has to be estimated and the estimate has to be applied to the time offset between server and client. If two-way time transfer is used (Fig. 1, right diagram), messages are exchanged mutually between a client and a server. This offers more information on the delay of the transfer messages (it is bounded by the round trip of the message exchange), and allows elimination of the delay dynamically, under the assumption that the network delay is symmetric for the two directions, i.e., that $\xi = 1/2$ in the figure.

2.2 Securing Time Synchronization Protocols

Since, for the most part, the content of time synchronization protocol packets is not secret, any form of confidentiality or secrecy is usually not considered a goal for any part of the communication (except for key exchange messages). A goal that is generally considered essential is packet integrity, linked with strong source authentication. Reference [16] contains a discussion about security goals in time synchronization contexts.

For securing unicast-type time synchronization content, most specifications use symmetric key cryptography. Some of them use standard shared key procedures, forcing the time server to keep an individual key for each client association. An example of this is the symmetric-key authentication procedure of the NTP, which was first defined for NTP Version 3 [13]. Other specifications mostly try to circumvent the need for the server to memorize the shared key, either by

enabling the server to regenerate the shared key [11,22] or by having the server encrypt its full association state and distribute it to the appropriate client [5]. Besides symmetric key techniques, external security measures such as MACsec, IPsec, and (D)TLS are candidates for securing time synchronization protocol packets [6,16]. It is also possible to use asymmetric cryptography for the creation of signatures for time synchronization traffic, although this comes at the cost of significant overhead and is therefore often excluded as a possibility [16]. In the remainder of this work, we forego further detail on securing unicast-type time synchronization traffic, since our focus is on attacking a particular scheme for securing broadcast-type time synchronization.

For securing broadcast-type time synchronization messages, the specifications generally use different techniques than those used in the unicast case. Although broadcast-type time synchronization might seem like an application for classical asymmetric cryptography, the computational cost is often an essential argument against it. Instead, specifications often apply the TESLA protocol [18] or one of its variants [8,17]. This class of specifications (the entirety of which we call "TESLA-like mechanisms") achieves asymmetric properties while using purely symmetric cryptography. They can be used for securing broadcast-type time synchronization either natively for a newly specified protocol [24,27] or as an addition to existing protocols [22]. Typically, the symmetric cryptographic measures are hash-based message authentication codes (MACs), which have a very low computational cost. The asymmetric properties are achieved by using a predefined schedule for usage and disclosure of keys: The sender attaches to each packet a MAC generated with a key that has not yet been disclosed and is thus known only to the sender itself at that point in time. A receiver buffers a packet for later validation of the included MAC. At a later time, the sender discloses the key, enabling the receivers to start the validation of the buffered MACs. For the scheme to work, it a receiver must be sure that the key used to generate a received packet has not been disclosed; otherwise, the MAC and therefore the whole packet may have been generated by an adversary. To this end, a predefined time schedule is used: time is partitioned into intervals in advance. Each time interval is associated with a key which is obtained in reverse order from a one-way key chain. For details on this, we recommend that the reader to either consult Reference [18] or study the way that keys are generated in the Minimal Example Protocol in Sect. 3. Because TESLA-like mechanisms are based on releasing messages on a pre-determined schedule, they require the client to have its clock loosely synchronized with the server's in order to establish the required security property. Loose synchronization is important as an initial requirement, which is typically met by performing time synchronization exchanges during the bootstrapping of the broadcast message stream. Such prior time synchronization exchanges are usually secured by means of methods other than the TESLA-like mechanism; these methods usually have a higher overhead, either computationally or in terms of communication bandwidth. However, the security of any TESLA-like scheme is based on the assumption that any client's clock is loosely synchronized to that of the server not only initially, but that it keeps the

necessary degree of synchronization throughout the whole communication. At the same time, the content of the messages that are secured directly influences the degree of synchronization of a client's clock to the server's clock. This creates a strong interdependency between clock synchronization and security, which is the basis for the attack described in Sect. 4.

3 Minimal Example Protocol

In this section, we define the Minimal Example Protocol (MEP), which is used for illustration in later sections. When correctly applied, it provides the clients with guarantees for packet integrity, linked with strong source authentication. Before we present the protocol's steps, we supply some essential protocol notation.

The agent names Alice, Bob, and Mallory are representative of a client, server, and attacker, respectively. In short form, the client, Alice, is denoted as A, the server, Bob, is denoted as B, and the attacker, Mallory, is denoted as M. The clock of a participant X is denoted by C_X and $C_X(t)$ denotes the value that clock reads at time t. Furthermore, the expression $\mathrm{Adj}(\mathrm{C_X}, \delta)$ denotes the process responsible for adjusting the clock C_X to compensate for a reported offset of amount δ from a reliable time synchronization source. The binary operator $||$ is used to represent the concatenation of messages. We assume that there is a fixed cryptographic hash function h that all participants have agreed on using. For a given key value K and message m, the expression $\mathrm{MAC}[K](m)$ stands for the keyed hash message authentication code using the hash function h mentioned above, computed over m and with K as the key.

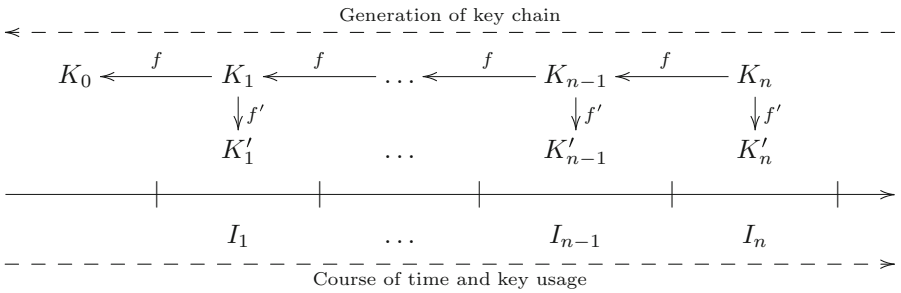

Fig. 2. Depiction of the generation of the one-way key chain for TESLA-like mechanisms.

Below, we present the protocol steps of the MEP, which help with the explanation of the attack in Sect. 4, and also serve to illustrate possible countermeasures in Sect. 5. We assume that Alice wants to synchronize her clock C_A to Bob's reference clock C_B. For simplification, we also assume that Bob's reference clock is perfect, i.e., that for any absolute time value t we have $C_B(t) = t$.

Additionally, we assume that for a chosen number n of intervals, Bob has generated a one-way key chain as follows (a graphic representation of this scheme can be seen in Fig. 2):

1. He has generated a key K_n randomly.
2. He has then applied a one-way function f to generate $K_i = f^{n-i}(K_n)$, for all values i with $0 \leq i \leq n-1$.
3. He has applied another one-way function f' to generate a chain of MAC keys $K'_i = f'(K_i)$, for $1 \leq i \leq n$.

Furthermore, we assume that Alice has received the following set of TESLA parameters in a way that guarantees that they are the same values that Bob uses:

- the starting time s_1 of the first interval I_1,
- the uniform duration L of all time intervals,
- the disclosure delay d, denoting the number of intervals between the usage and disclosure of a key in the chain,
- the base key K_0 for the one-way key chain, and
- the one-way functions f and f'.

Additionally, we assume that there is a constant delay Δ for messages traveling from Bob to Alice and that Alice has precisely estimated Δ.[1]

We define the MEP as follows: The time server, Bob, sends a broadcast-type packet P_i at the starting time s_i of each interval I_i. Such a packet is constructed by defining $P_i = i \mid\mid C_B(s_i) \mid\mid \mathrm{MAC}[K'_i](C_B(s_i)) \mid\mid K_{i-d}$, where for all cases $i - d \leq 0$, a predefined "empty" value is used instead of K_{i-d}.

When Alice receives P_i at time r_i, she first checks the timeliness of P_i. In the MEP, she does this by simply checking the inequality $C_A(r_i) - \Delta < s_{i+1}$, in which $C_A(r_i) - \Delta$ represents the assumed sending time of P_i. If Alice has verified the timeliness of a packet P_i, she saves P_i together with the value $C_A(r_i)$ of her clock at the reception time of P_i. An additional action that Alice takes upon receipt of any packet P_i is to check whether the disclosed key K_{i-d} is a valid key (whenever it is not the empty value). She can do this by using the one-way function f to check K_{i-d} against an already verified key, for example K_0 (in this example, she verifies the equality $K_0 = f^{i-d}(K_{i-d})$). When Alice has verified a key K_{i-d}, she can then use it to derive K'_{i-d}. With this, she can try and verify the integrity of P_{i-d} (recall that P_{i-d}, together with the timestamp value $C_A(r_{i-d})$, was stored beforehand, given that its timeliness was verified at reception time). For the verification of a packet P_{i-d}'s integrity, Alice simply verifies that her calculation of the message authentication code $\mathrm{MAC}[K'_{i-d}](C_B(s_i))$ agrees with the MAC value included in P_{i-d}. If Alice has verified the integrity of a packet P_{i-d}, she calculates the difference $\delta_{i-d} = C_A(r_{i-d}) - (C_B(s_{i-d}) + \Delta)$ and then starts the process $\mathrm{Adj}(C_A, \delta_{i-d})$. For the purpose of this minimal example protocol, we assume that $\mathrm{Adj}(C_A, \delta_{i-d})$ simply sets the clock C_A back by δ_{i-d}.

[1] To model Δ as factually constant in the network simplifies the analysis. Assuming that Alice treats it as constant makes sense because, as long as she only has one-way time synchronization communication data available, she cannot reliably determine or compensate for varying network delays.

4 The Attack Vector

Before we go on to describe the attack, we first present the attacker model. We assume that there is exactly one attacker[2] (Mallory) and that she complies with the Dolev-Yao attacker model [4]. This gives her the following capabilities:

- She can overhear and intercept any message that is sent on the network. In particular, she can prevent delivery of any message in its original form.
- She can synthesize messages by inventing new values (secret keys or nonces are assumed to be unguessable), by assembling tuples from known values, by disassembling known tuples into their components, and by using any operator with any values (including keys) as long as they are in her knowledge.
- She can send messages to any agent on the network, pretending to possess any identity she chooses. However, it is still possible to verify authorship of a message by cryptographic means, through appropriate use of secrets.
- One possible combined application of the abilities mentioned above is that Mallory can delay the delivery of a message by first preventing it from being delivered and later replaying it. This possibility should be highlighted in the context of time synchronization, since performing this technique (called "delay attack" or "pulse delay attack") can degrade the performance of time synchronization and is very simple to perform [7,16].

It should be noted that the Dolev-Yao attacker model is very permissive, much more so than attacker models used elsewhere, for example in the recent work [3] about attacks on a specific implementation of the NTP protocol. The attacker model was chosen to account for the generic applicability of the attack vector, as well as to accommodate the fact that we intended to do a formal analysis with a model checker such as UPPAAL [2]. Thus, our model is susceptible to the well-known state-explosion problem; in our experience, modeling more aspects of the network, to say nothing of cryptographic mechanisms, greatly increases the state space size.

In order to successfully perform the attack, Mallory performs the following phases. Phase 1 aims to cause enough of an offset between Alice's and Bob's clocks that it is possible for Alice to believe that a key is still undisclosed, while in reality, Bob has already disclosed it. Phase 1 comprises several steps:

1. Mallory starts by choosing an interval i_1 and by consistently delaying packets from Bob's TESLA-secured broadcast stream, starting with P_{i_1}. She delays them by a delay d_1 such that Alice still accepts them as timely, i.e., such that $C_A(s_{i_1} + \Delta + d_1) \in I_{i_1}$.
2. Mallory continues this until the delaying of these packets has taken an effect on Alice's clock (this will take at least until the key for P_{i_1} has been disclosed and this packet has been successfully verified). The expected effect is to set Alice's clock back by an additional delay $d_2 > 0$.

[2] This assumption is made for simplification. The assumed situation is equivalent to a situation where several attackers are cooperating, or to a situation where one attacker is being helped by one or more dishonest protocol participants [25].

3. After the effect of the first delay has appeared, Mallory delays another stream of packets, beginning with P_{i_2}. Due to adjustments to Alice's clock because of the delay added to the time synchronization packets $P_{i_1}, P_{i_1+1}, \ldots, P_{i_2-1}$, the timeframe in which Alice will accept P_{i_2} as timely has increased by d_2. Hence, Mallory is able to delay the stream of packets beginning with P_{i_2} by an amount $d_1 + d_2$, which is strictly greater than d_1.[3]
4. The procedure described in Step 3 is iterated further, resulting in ever larger possible delays. These delays in turn lead to ever larger offsets between Alice's and Bob's clocks, and therefore to ever larger timeframes during which Alice still accepts packets as timely. At some point, the offset surpasses the value $(d-1) \cdot L$, where L is the length of the time intervals, and d is the disclosure delay as defined for the TESLA scheme.

When the offset between Alice's and Bob's clocks is larger than $(d-1) \cdot L$, Phase 1 is finished, and Mallory switches to Phase 2 of the attack. There is now sufficient time to intercept a packet using a disclosed key from Bob, to forge a packet based on that key, and to relay the forged packet to Alice fast enough that Alice still accepts it as timely. Using this technique, Mallory is now able to successfully pretend to be Alice's time server, Bob. The security gained by using the TESLA protocol on the time synchronization traffic is therefore compromised.

We now go through the procedure of the attack, supplying specifics for an application of it to the MEP, where $d = 2$ is chosen for simplicity. We start with the steps for Phase 1 (see also Fig. 3 for an illustration).

- For Step 1, Mallory starts with the packet P_1, delaying it by $d_1 = \frac{2}{3} \cdot L$.
- For Step 2, she continues this for P_2 and P_3. This triggers an effect upon the arrival of P_3: Alice extracts K_1, successfully validates it, derives K_1', and uses it to successfully validate the MAC included in P_1. As a consequence, she sets her clock C_A back by the amount $d_2 = d_1 = \frac{2}{3} \cdot L$.
- For Step 3, Mallory delays the packets starting with P_4 by the increased amount $d_1 + d_2 = \frac{4}{3} \cdot L$. It should be noted that, because C_A was set back, Alice still accepts these packets as timely, even though they arrive more than one interval length after their sending time.
- Step 4 is conveniently short in our given example, as the offset surpasses the value $(d-1)\,L = L$ even after the arrival of P_6.

For Phase 2, Mallory can use the resulting overlap intervals in which Bob's and Alice's clocks have an offset of more than one interval. She can intercept all packets starting from P_7, blocking them from being delivered. When Bob sends P_9 (which includes K_7), Alice still believes to be in time interval I_7. At this point, Mallory can read K_7, derive K_7' and invent a bogus packet Q_7, complete with a valid MAC using K_7' as its key. She has a timeframe of $\frac{2}{3} \cdot L$ to do this and deliver Q_7 to Alice, who will then still accept it as timely. If she keeps this technique up for P_8 and P_9, she can disclose the intercepted (correct) key K_7

[3] Note that the value of d_2 is unknown to Mallory. However, she is able to estimate it from her knowledge of the time synchronization mechanism.

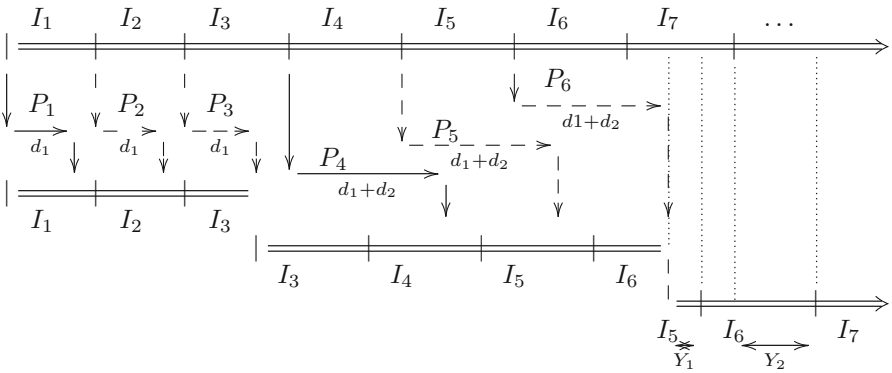

Fig. 3. Schematic description of Phase 1 of the attack, with $d = 2$ chosen.

to Alice in a believable way. Alice will then validate the bogus packet Q_7 and adjust her clock according to the bogus time values that it might carry.

The attack relies purely on the interdependency of clocks and cryptography that is specific to time synchronization protocols which are secured with TESLA-like mechanisms. There are no weaknesses in the preparation stage needed for the attack to work. In particular, it can be assumed that the client, Alice, and the server, Bob, have clocks which are initially synchronized to a specified degree; the attack works even if this initial synchronization is impossible to disrupt. Also, it can be assumed that the broadcast schedule for a TESLA-like mechanism (Reference [18] provides some detail) has been exchanged securely; the attack works even if this exchange is impossible to disrupt.

Note that, in the MEP, we chose the mechanism of bluntly "hard-setting" the absolute value for the actual adjustment of the clock mostly for its simplicity of presentation. Many time synchronization protocols will use a more refined mechanism. For example, the NTP will try to make clock adjustments using only frequency corrections, using increased frequency if the clock is behind its reference clock and decreased frequency if it is ahead. It will set the absolute clock value only if the network communication implies large offsets persistently for a long period of time [14,15]. In addition to the differences between how protocol specifications describe clock adjustment, some specifications (such as the one for PTP) only provide abstract concepts for reading and setting clocks, leaving the technical details open. In such cases, the specific technical processes for clock adjustment depend on the particular implementation. However, using means of clock adjustment other than hard-setting or having restrictions on when hard-setting may occur can only delay the effect of an attacker's manipulation for a certain amount of time. Eventually, even large offsets will always be corrected if they are reported persistently (see Option 5 in Sect. 5 and the related discussions).

5 Discussion of Countermeasures

We now look at methods which might mitigate or counter the attack described in Sect. 4. Let us first consider techniques which do not change the protocol itself, but some aspect of the channel(s) via which it is used.

Option 1: Alice can try to mitigate her vulnerability to the attack by selecting multiple channels via which she synchronizes her clock. The approaches in this direction range from just picking multiple time servers [12] to using multiple network paths in order to reach the same time server via different channels [21].

A disadvantage of Option 1 is that Mallory "only" needs to perform the same attack on all the channels via which Alice synchronizes her clock to a time source. However, the difficulty of Mallory's task is proportional to the number of channels Alice uses; in practice, this might prevent Mallory from successfully completing the attack. An advantage of Option 1 worth mentioning is that it is very easy to implement in a modular fashion: the respective secured time synchronization protocol simply needs to be instantiated multiple times and run via the different channels.

Option 2: Another way of defending against the attack is to enforce the confidentiality of the time synchronization traffic, e.g. by means of full encryption of all packets (see, for example, Sect. 5.8 of [16] for some discussion about confidentiality in the context of time synchronization).

Ideally, if Mallory cannot identify the packets she needs to delay, she cannot perform Phase 1 of the attack. Additionally, she may also not be able to execute Phase 2 under perfect confidentiality, because she would be unable to extract Bob's disclosed keys. However, it should be noted that keeping time synchronization traffic confidential is not as simple as merely encrypting the packets, as metadata already provides a great deal of information and Mallory can mount attacks even with incomplete information. Additionally, confidentiality in a broadcast setting would require either a group key solution or asymmetric cryptography. A group key approach is ineffective if Mallory finds a way to join the group. Asymmetric cryptography contradicts the requirement of low computational cost, which is the main advantage of employing TESLA-like mechanisms.

In contrast to Options 1 and 2, which work purely by changing the time synchronization protocol's underlying channel(s), the options below employ modifications to the protocol message flow to defend against the attack.

Option 3: One way of employing changes to the protocol flow is to include a cryptographically secured unicast exchange between Alice and Bob in order for Alice to ask, explicitly or implicitly, whether a certain key from the TESLA key chain has already been disclosed. We call this a *timeliness confirmation exchange*.

As a minimum, the exchange should be a two-way transmission initiated by Alice. For an overview of how such an exchange might work, we consider the following example: Alice's timeliness confirmation request TC_A consists of a value indicating that she wants to ask about the key K_i (in this case, she could just send i) and Bob's response TC_B also consists of this value in the case where the key is yet undisclosed and consists of a standard value of -1 otherwise. The desired outcome here is that Alice gets a guarantee that, at a time later than t_3, a given key K_i has not yet been disclosed. If this holds true, she can deduce that, at the reception time t_2 of P_i, the key had also not yet been disclosed, meaning that the packet arrived in a timely fashion. This example shows the main advantage of the option discussed; including a timeliness confirmation exchange breaks down the question of broadcast packet timeliness to a pure ordering of messages, which can be judged by the client without additional assumptions. Thus, the timeliness confirmation exchange enables the client to verify timeliness without even referring to a local clock. This prevents Phase 2 of the attack from being executable. There are also disadvantages to such an exchange. In most contexts, adding a secured unicast message exchange would defy the purpose of using a TESLA-like mechanism in the first instance, because it removes one of the key advantages of TESLA-like mechanisms – specifically, that a key exchange for each server-client association is not required. However, as with Option 4, it should be mentioned that specifications may already support secure unicast communication for other purposes, in which case the addition of a unicast exchange would not be as costly. Furthermore, the resulting message exchange pattern fits very well with some existing time synchronization protocols, in particular with the Precision Time Protocol (PTP) [9].

Option 4: Another way of changing the protocol flow as a defense is to regularly request time synchronization from the server via alternative communication, which has to be secured in order to provide additional reliability.

Such auxiliary time synchronization mitigates the effect of Phase 1 of the attack because the gradually introduced offset is negated. Most specifications that rely on TESLA-like mechanisms have some method of achieving initial time synchronization, which can be applied as an alternative communication channel.

Option 5: In order to mitigate the attack, it also helps if regulations are enforced for the clock adjustment mechanism that limit the amount of offset that can be maliciously introduced during Phase 1. Plausibility checks can be used for this purpose, as well as ignoring measured offsets that are above the specified upper bounds.

By itself, this option can only delay the time by which Mallory is able to switch from Phase 1 to Phase 2. It can, however, keep a TESLA-like mechanism provably secure over a certain period of time. This option can therefore also be used to more efficiently utilize other options, namely Option 4 or 3, as it provides better guidelines for the frequency in which these options need to be applied. To explore this issue further, we introduce a parameter $D_{\max}$ which represents

the maximum amount of offset that Mallory can additionally introduce over the course of one interval by using delay attacks as described in Phase 1 of the attack. This parameter is assumed to be simplified in the sense that it is already adjusted for values like the existing offset between the participants' clocks before the interval in question (caused by Mallory or other influences), the frequency error of Alice's clock, and fluctuations in the network delay. Under this assumption, we get inequalities $0 < D_{\max} \leq L$.

A disadvantage of both Options 3 and 4 is that inserting auxiliary communication into a broadcast-based protocol might represent a significant change to the communication model, making these options significantly more intricate to adopt than Option 1 or 2. On the other hand, Options 3 and 4 have the advantage that they can provide complete protection from Phase 2 of the attack described in Sect. 4. For this purpose, it is necessary to additionally regulate the configuration values (specifically the number of intervals and interval length of the TESLA-like mechanism, as well as aspects of clock adjustment as discussed under Option 5) of the employed protocol in such a way that it makes reaching Phase 2 of the attack very difficult or even impossible.

For the automated analysis presented in Sect. 6, we have chosen to include Options 3 and 5 in the model. The model could easily be adjusted to allow Option 1 to be included as well, but we did not pursue this path because we found the security gains for that option to be much harder to quantify, and decided it was not worth the additional state space growth. Modeling Option 2 went against our decision to eliminate cryptographic aspects from the model. Option 3 is not included in the model because, for the state space growth it causes, its practical relevance is doubtful due to the fact that, if a protocol does not already include a timeliness confirmation exchange, it takes a great deal of effort to integrate it retroactively.

6 The UPPAAL Model

In this section, we present an automated analysis of the attack applied against the MEP. The analysis was performed with UPPAAL [2], a model checker based on the theory of timed automata. UPPAAL models a system as a network of such automata running in parallel, with some additional features added to its modeling language such as bounded integer variables that are part of the system state. It enables users to check properties specified in a query language that is a simplified version of TCTL (Timed Computation Tree Logic [1]). The obtained results are, as of yet, available only for undesirably large ratios of small-step delays to interval lengths (currently, a ratio of $1/9$ was achievable on our main machine). However, on the obtainable scale, the results support that the attack is performable under the right parameters; they also support that a combination of Options 4 and 5 from the possible countermeasures listed in Sect. 5 does in fact protect against the attack if the parameters are chosen carefully. Phase 1 of the attack does not require the insertion of false or modified messages, nor does it depend on the cryptography applied. Instead, it uses only timing-dependent

attacks, enabling the attacker to circumvent cryptographic security completely in Phase 2. This fact is one reason for the choice of UPPAAL as the automated tool for the analysis: UPPAAL is highly able to deal with timing-related questions. Furthermore, this fact is the justification for the use of a number of abstractions and simplifications to specifically tailor the model to the attack described in Sect. 4, focusing the analysis on the switch from Phase 1 to Phase 2. This was done in order to keep the state space smaller. The first important resulting simplification is that the model does not take Mallory's capabilities of inserting false messages into account. The second is the exclusion of any cryptographic functions from the model. An additional simplification to the model is that the client's clock value is modeled as the drift from the server's clock, so that its range is not directly proportional to longer run times of the system.

Our UPPAAL model is a version of the MEP that is extended with measures corresponding to countermeasure Options 3 and 5. The system models one server and one client (the number of clients could easily be made configurable at the expense of significant state space growth), as well as the attacker's capability of delaying a packet's delivery. The model ignores all cryptographic aspects of the MEP or its extensions, such as the one-way functions f and f', as well as the chain of keys K_i and even the MAC function. It instead assumes that these aspects work as described and only treats the other aspects, namely the participants' clocks C_B and C_A (more accurately, the drift $C_A - C_B$ is modeled), the adjustment process $\mathrm{Adj}(C_A, \delta)$ as well as a number of configurable parameters. These parameters include the interval length L, the disclosure delay d, the assumed constant network delay Δ, the maximum number $u_{\max}$ of unicast repetitions, the maximum number $n_{\max}$ of broadcast repetitions between unicast repetitions, and finally the parameter $D_{\max}$ introduced in Sect. 5. The model is split into nine separate automata: a very simple one for advancing the clock values, one for each of the participants' clocks, one for each of the participants' behavior models (in the server's case there are two of these, one for broadcast and one for unicast) and one for each of the three existing message types: unicast time request, unicast time response, and the broadcast time message. To review our UPPAAL model in detail, please download its source code[4].

The first goal of the UPPAAL analysis was to show that there exist conditions under which the attack is feasible against the MEP. The second goal was to show that a combination of countermeasures (Options 4 and 5 in particular) can protect the MEP from the attack. The main queries we evaluated for different parameter sets were the following (where $\mathrm{I}(X)$ represents the number of the interval that participant X believes themselves to be in and where j represents the number of intervals that the protocol has been running for):

$$A\Box\ \mathrm{I}(A) \geq \mathrm{I}(B) - d + 1, \tag{1}$$

$$E\Diamond\ j = {}^{(d-1)\cdot L}/_{D_{\max}} + d \wedge \mathrm{I}(A) \leq \mathrm{I}(B) - d, \tag{2}$$

[4] The UPPAAL source code is available for download here: http://www8.cs.fau.de/~milius/UPPAAL%20Model%20(TESLA-Like%20Mechanisms).zip.

$$A\Box\, j < {}^{(d-1)\cdot L}/_{D_{\max}} + d \implies \mathrm{I}(A) > \mathrm{I}(B) - d. \tag{3}$$

Informally, the queries can be read as follows:

- Query 1: "The interval Alice believes herself to be in is always at most $d-1$ behind that which Bob is in."
- Query 2: "There is a state in which the interval that Alice believes herself to be in is at least d behind that which Bob is in and this state happens after at least ${}^{(d-1)\cdot L}/_{D_{\max}} + d$ intervals have passed."
- Query 3: "For all states in which less than ${}^{(d-1)\cdot L}/_{D_{\max}} + d$ intervals have passed, the interval that Alice believes herself to be in is at most $d-1$ behind that which Bob is in."

For $n_{\max} < {}^{(d-1)\cdot L}/_{D_{\max}} + d$, Queries 1 and 3 were always affirmed if the check was completed, while Query 2 was always (trivially) negated. For $n_{\max} \geq {}^{(d-1)\cdot L}/_{D_{\max}} + d$, on the other hand, Query 1 was always negated if the check was completed, while Queries 2 and 3 were always affirmed. This implies that Phase 1 of the attack can be completed in the model if and only if the protocol runs for at least ${}^{(d-1)\cdot L}/_{D_{\max}} + d$ intervals and that this represents a sharp bound. On the one hand, these results affirm that the attack is feasible against the unmodified MEP. On the other hand, they also affirm that a combination of countermeasures 4 and 5 can protect against the attack if the combination of $n_{\max}$, L and $D_{\max}$ is chosen correctly.

The checks were performed on a computer running a 64-bit version of Windows 7 on an Intel i5 dual core at 2.6 GHz, with 8 GB of RAM. To date, we have only been able to run the command line verifier tool of UPPAAL under Windows, where it can apparently use only 2 GB of RAM. Therefore, 2 GB of RAM represents a bottleneck for our checks under the requirement of precise runtime and memory usage logs. Under these conditions, the $1/9$ ratio of $D_{\max}$ to L was the best that allowed all query checks to finish. Performance data can be seen in Fig. 4. The relevance of the analysis of the protocol to practical applications will increase depending on how small the ratio of small-step delays to interval lengths can be made. We are working on refining the model further in order to obtain better results in this area; an attempt to also model the server's clock to be more independent from the overall run time of the system is among the next measures to be tried in order to improve the ratio.

7 Evaluation of Existing Specifications

We discuss three specifications that use TESLA for securing broadcast-type time synchronization traffic: NTS-secured NTP [23], TinySeRSync [24], and ASTS [27]. Without providing detailed descriptions of these protocols, we provide a sketch of how TESLA-like mechanisms are employed and to what extent they might be susceptible to the attack described in Sect. 4. For all three specifications, we constructed scenarios with certain settings that make them robust

Query	Ratio	Runtime in s	Memory Usage in MB
1		29.95	134
2	1/6	27.14	244
3		29.14	271
1		211.37	713
2	1/9	192.77	1,520
3		196.05	1,586
1		286.59	1,091
2	1/10	236.97	Out of memory
3		233.14	Out of memory

Fig. 4. The performance data of our UPPAAL model on the computer used

against the attack in the sense that Phase 2 is completely excluded. For NTS-secured NTP and for ASTS, we also constructed scenarios in which the specifications are vulnerable to the attack in the sense that Phase 2 can be executed against them. The parameter spaces between the given scenarios are the subject of future research. Note that our discussion of those scenarios represents only initial assessments derived from the respective specification documents.

NTS-Secured NTP: The Network Time Security (NTS) specification aims to secure time synchronization in packet-switched networks. The project is motivated by the fact that neither of the predominant time synchronization protocols – NTP and PTP – currently provide adequate security mechanisms (Reference [20], for example, provides an analysis of the weaknesses in NTP's Autokey protocol [11]). Currently, the NTS specification is still in the standardization process at the IETF [22]. Here, we focus on the application of NTS to the NTP, since this is already specified in an additional draft document [23]. Time synchronization in NTP's unicast mode is secured via a unique shared secret between client and server, which is specified in such a way that the server is able to regenerate it on request, thus preserving server-side statelessness. NTP's broadcast mode is secured via TESLA [22, Sect. 5 and Appendix B of Version 08], [23, Sects. 4.2, 5.1.2 and 5.2.2 of Version 00].

The fact that NTS-secured NTP offers secured unicast time synchronization enables a defense against the attack in the sense of Option 4. The specification mentions that secured unicast messages are used to set up the initial time synchronization required for the TESLA-like mechanism, but does not mention whether or how often unicast exchanges should be used after this initialization. Since Draft Version 05, the NTS specification has mentioned "keycheck" message exchanges for broadcast messages. These represent timeliness confirmation exchanges as discussed in Option 3. It should also be noted that, for offsets less than 128 ms, the NTP avoids setting the client's clock but adjusts its frequency in order to minimize the time offset [15]. Only if offsets of more than 128 ms are reported consistently for a period of over 15 min will the protocol set the time of the client's clock. This represents a countermeasure in accordance

with Option 5. A combination of these countermeasures enables us to construct sets of configuration values with which NTS-secured NTP is completely protected against Phase 2 of the attack. For the first scenario, we assume that Alice performs a keycheck exchange after the receipt of each broadcast packet. This implies that Phase 2 of the attack can never work, because Alice will not accept a non-timely packet as eligible, independently of the offset between her clock and Bob's clock. However, this scenario contradicts the principle of broadcast communication and is therefore not feasible in real-world applications. For the second scenario, we assume the following set of parameters: $d = 2$, $L = 64\,\mathrm{s}$, and $n = 14$. Furthermore, we assume that secure time synchronization via unicast message exchanges is an inherent part of the re-initialization process of the TESLA-like mechanism. Here, we make reference again to the regulation that discards any offset over 128 ms unless it is reported consistently for over 15 min and note that, with these values, the TESLA-like mechanism runs for 896 s, which is just under 15 min. Therefore, all offsets over 128 ms will be ignored during the broadcast synchronization, which yields $D_{\max} = 128\,\mathrm{ms}$, even if all other circumstances allow for a higher value of $D_{\max}$. Under these conditions, an upper bound for the malicious offset that Mallory can accomplish during Phase 1 of the attack is $14 \cdot 128\,\mathrm{ms} = 1792\,\mathrm{ms}$, which is far below the value $(d-1)\,L = 64\,\mathrm{s}$ needed to start Phase 2. This scenario therefore offers complete protection against Phase 2 of the attack, even without the use of keycheck message exchanges.

We also construct a scenario in which NTS-secured NTP is vulnerable to the attack. For this scenario, we choose $d = 2$, and then choose $L = 512\,\mathrm{s}$ and $n = 100$. We also assume $D_{\max} = 0.75$ and $L = 384\,\mathrm{s}$. Under these conditions, any offset that Mallory introduces for two consecutive broadcast messages is therefore reported for 1024 s, which is over 15 min.

TinySeRSync: The TinySeRSync protocol [24] is intended as a secure and resilient time synchronization subsystem, with particular application in networks of wireless sensors running TinyOS. The TinySeRSync protocol secures time synchronization traffic via two tasks (its authors use the term "phases") running in parallel; each of them is run periodically, with no predefined interdependence between their frequencies. The tasks start in order (task one first, task two only after the completion of task one) and then run independently, without communicating or synchronizing with each other directly. They only interact by manipulating the same clock and network connections. The first task has every pair of neighboring nodes in the assumed sensor network perform secure, single-hop pairwise synchronization. This is a one-way, unicast-type time synchronization that is secured by MACs (the specification uses the term "message integrity code"). The MACs are generated with shared secret keys, requiring every pair of neighbors to perform an appropriate secure key exchange as part of the network preparations. The second task employs broadcast-type time synchronization. This is secured via μTESLA [19, Sect. 5], a variant of TESLA specifically designed to be slim enough to work in sensor networks. As already mentioned, the two tasks are run in parallel and periodically. The unicast-type messages from task one, which is run periodically, constitute a countermeasure

against the attack in accordance with Option 4. Consequently, the transition from Phase 1 to Phase 2 of the attack has to be reached between two executions of task one. However, it is stated that task one is typically run at a higher frequency than task two [24, Sect. 7.1]. Therefore, the transition would have to be made after, at most, one execution of task two, which is not possible. As such, TinySeRSync can be expected to be immune against Phase 2 of the attack described in Sect. 4, given that the configuration values, in particular for the periodicity of the two tasks, follow the recommendations given in Reference [24].

ASTS: The authors of ASTS [27] compare their proposed protocol with TinySeRSync, highlighting its comparably higher accuracy as well as the fact that ASTS is more lightweight than TinySeRSync. Both of these properties are achieved by dropping the mechanisms of TinySeRSync's first task, eliminating the secure single-hop pairwise synchronization. The requirement made by μTESLA for initial time synchronization is fulfilled by using a global group key mechanism (instead of μTESLA) for the first period of broadcast-type one-way time synchronization. After this, ASTS employs μTESLA exclusively for securing the broadcast-type one-way time synchronization messages in subsequent synchronization periods.

Disregarding the question of the extent to which the group key mechanism might open the protocol up to internal attackers in the first period, we focus on the usage of the TESLA-like mechanism after the first period. The authors of ASTS suggest using an estimated value for packet propagation delay that is negligible against the length of a μTESLA interval, as well as using this scheme for a significant amount of intervals with low disclosure delay (parameters used for the performance analysis environment are $d = 2$, interval length $L = 60\,\mathrm{s}$ for a number of $n = 10$ intervals [27, Section 3.A]). These settings appear to enable the attack described in Sect. 4 in principle, although its practical feasibility will depend on data we could not obtain from the paper: the maximum introducible delay $\leq D_{\max}$ (which depends on the chosen value for the propagation delay estimate parameter, as well as on the point in an interval that the nodes send out their packets and the details of the employed clock adjustment mechanism) and, finally, the actions that are taken when μTESLA is re-initialized. We focus on the last item and use it to construct a scenario wherein ASTS is vulnerable to the attack. If re-initialization of μTESLA does not entail any communication related to Options 4 or 3 (it appears from Reference [27] that it does not), then we can treat this as if μTESLA is running for an unbounded number of intervals. In this case, any measures related to Option 5 or Option 1 can at best provide mitigation against the attack in the sense that it takes longer to switch from Phase 1 to Phase 2, but such measures cannot prevent the attack indefinitely.

If, on the other hand, ASTS is configured to include time synchronization via the global group key mechanism as part of re-initialization of μTESLA, then this constitutes a countermeasure conforming to Option 4. In this case, Phase 2 of the attack has to be started during the course of one run of μTESLA. For this to be possible even in principle, Mallory needs $D_{\max} > \frac{(d-1)\cdot L}{n-1}$ to hold, which translates to $D_{\max} > \frac{60}{9}\,\mathrm{s} = 6.\overline{6}\,\mathrm{s}$ in the performance analysis environment.

Configuring Bob in such a way that he only sends broadcast packets after 55 s of an interval have passed, we then achieve $D_{\max} \leq 5\,\mathrm{s}$. Thus, in this scenario, ASTS is robust against the attack in the sense that Phase 2 of it is unreachable.

8 Conclusions

In this paper, we have demonstrated an attack vector that can be used by an adversary in order to break the security of broadcast time synchronization protocols which are protected by TESLA-like mechanisms. The adversary makes use of very specific interdependence between timing and security occuring under such circumstances.

We have provided options for countermeasures and discussed their respective merits. Furthermore, we have provided insight into how some of these options can be combined to defend against the attack, in particular against its Phase 2. We have performed an automated analysis by means of the UPPAAL model checker. This analysis has confirmed the vulnerability of the unmodified MEP model to the attack; it has also confirmed the effectiveness of a combination of countermeasures. Next, we have evaluated three existing protocols and found that all of them provide mechanisms that correspond to one or more of the options presented and are suitable to defend against the attack. In all of the cases, however, the protection achieved depends on how these mechanisms are applied, specifically the exact configuration values that the protocol uses. It is important for the developers and users of these protocols to be aware of the attack vector so that it can be defended against in existing environments.

Our next goal is the refinement of our UPPAAL model to allow for better ratios of maximum accepted offsets to interval length. Additionally, a project is underway in which an implementation of either TinySeRSync, ASTS, or both is to be subjected to an attack according to the attack vector discovered. This attack is intended to be performed by a piece of attacker software either in a controlled real network or in a simulated environment. The objective of this work is to prove practical relevance for existing specifications, and further quantify the vulnerability or security of implementations under given circumstances.

Future research should additionally include in-depth analyses with regard to the attack outlined in this paper of any (current and upcoming) specifications which use TESLA-like mechanisms to secure broadcast-type time synchronization communication. Potential applications include a finished version of NTS-Secured NTP as well as the ongoing work in the area of adding security mechanisms to the IEEE 1588 standard (PTP). A more remote, yet interesting question that could be pursued is the extent to which an attacker can make use of disturbances via delay attacks which only trigger frequency manipulation. Is it possible to manipulate the frequency of a client's clock with consequences as severe as Phase 2 of the attack described in Sect. 4? Another question is whether it is possible to find attack vectors which use delay attacks to break the security of synchronization processes with unicast messages (as opposed to broadcast messages).

References

1. Alur, R., Courcoubetis, C., Dill, D.: Model-checking for real-time systems. In: Proceedings of Fifth Annual IEEE Symposium on Logic in Computer Science, LICS 1990, pp. 414–425, June 1990
2. Behrmann, G., David, A., Larsen, K.G.: A tutorial on Uppaal. In: Bernardo, M., Corradini, F. (eds.) SFM-RT 2004. LNCS, vol. 3185, pp. 200–236. Springer, Heidelberg (2004). doi:10.1007/978-3-540-30080-9_7
3. Brakke, E., Cohen, I.E., Goldberg, S., Malhotra, A.: Attacking the network time protocol. In: Network and Distributed System Security Symposium (NDSS), February 2016
4. Dolev, D., Yao, A.: On the security of public key protocols. IEEE Trans. Inf. Theory **IT–29**, 198–208 (1983)
5. Dowling, B., Stebila, D., Zaverucha, G.: Authenticated network time synchronization. Cryptology ePrint Archive, Report 2015/171 (2015). http://eprint.iacr.org/2015/171
6. Floeter, R.: Authenticated TLS constraints in ntpd(8). Web Post, February 2015. https://marc.info/?l=openbsd-tech&m=142356166731390&w=2
7. Ganeriwal, S., Pöpper, C., Capkun, S., Srivastava, M.B.: Secure time synchronization in sensor networks (E-SPS). In: Proceedings of 2005 ACM Workshop on Wireless Security (WiSe 2005), pp. 97–106, September 2005
8. Hu, X., Feng, R.: Message broadcast authentication in μTESLA based on double filtering mechanism. In: 2011 International Conference on Internet Technology and Applications (iTAP), pp. 1–4 (2011). http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6006446
9. Lee, K., Eidson, J.: IEEE-1588 standard for a precision clock synchronization protocol for networked measurement and control systems. In: 34th Annual Precise Time and Time Interval (PTTI) Meeting, pp. 98–105 (2002)
10. Liu, D., Ning, P.: Multilevel μTESLA: broadcast authentication for distributed sensor networks. ACM Trans. Embed. Comput. Syst. **3**(4), 800–836 (2004). http://dl.acm.org/citation.cfm?doid=1027794.1027800
11. Mills, D., Haberman, B.: Network time protocol version 4: autokey specification. RFC 5906, IETF Secretariat, June 2010. https://tools.ietf.org/html/rfc5906
12. Mills, D., Martin, J., Burbank, J., Kasch, W.: Network time protocol version 4: protocol and algorithms specification. RFC 5905, IETF Secretariat, June 2010. https://tools.ietf.org/html/rfc5905
13. Mills, D.L.: Network time protocol (version 3): specification, implementation and analysis. RFC 1305, IETF Secretariat, March 1992. https://tools.ietf.org/html/rfc1305
14. Mills, D.L.: Computer Network Time Synchronization: The Network Time Protocol. CRC Press, Boca Raton (2006). http://www.loc.gov/catdir/enhancements/fy0664/2005056889-d.html
15. Mills, D.L., Acm, M.: Adaptive hybrid clock discipline algorithm for the network time protocol. IEEE/ACM Trans. Networking **6**, 505–514 (1998)
16. Mizrahi, T.: Security Requirements of Time Protocols in Packet Switched Networks (2014). http://www.rfc-editor.org/rfc/pdfrfc/rfc7384.txt.pdf
17. Na, R., Hori, Y.: DoS attack-tolerant TESLA-based broadcast authentication protocol in Internet of Things. In: 2012 International Conference on Selected Topics in Mobile and Wireless Networking (iCOST), pp. 60–65, June 2012. http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6271291

18. Perrig, A., Song, D., Canetti, R., Tygar, J.D., Briscoe, B.: Timed efficient stream loss-tolerant authentication (TESLA): multicast source authentication transform introduction. RFC 4082, IETF Secretariat, June 2005. https://tools.ietf.org/html/rfc4082
19. Perrig, A., Szewczyk, R., Wen, V., Culler, D., Tygar, J.D.: SPINS: security protocols for sensor networks. In: Wireless Networks, pp. 189–199 (2001)
20. Roettger, S.: Analysis of the NTP autokey procedures. Web Publication of Project Thesis, February 2012. http://zero-entropy.de/autokey_analysis.pdf
21. Shpiner, A., Revah, Y., Mizrahi, T.: Multi-path time protocols. In: 2013 International IEEE Symposium on Precision Clock Synchronization for Measurement Control and Communication (ISPCS), pp. 1–6, September 2013
22. Sibold, D., Teichel, K., Roettger, S.: Network time security, July 2013. https://datatracker.ietf.org/doc/draft-ietf-ntp-network-time-security/
23. Sibold, D., Teichel, K., Roettger, S.: Using the network time security specification to secure the network time protocol, March 2015. https://datatracker.ietf.org/doc/draft-ietf-ntp-using-nts-for-ntp/
24. 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, CCS 2006, NY, USA, pp. 264–277 (2006). http://dl.acm.org/citation.cfm?doid=1180405.1180439
25. Syverson, P., Meadows, C., Cervesato, I.: Dolev-Yao is no better than Machivelli. In: First Workshop on Issues in the Theory of Security - WITS 2000, pp. 87–92 (2000)
26. Weiss, M.A., Eidson, J., Barry, C., Broman, D., Iannucci, B., Lee, E.A., Stanton, S.K., Goldin, L.: Time-aware applications, computers, and communication systems (TAACCS). NIST Technical note 1867, February 2015
27. Xianglan, Y., Wangdong, Q., Fei, F.: ASTS: an agile secure time synchronization protocol for wireless sensor networks. In: International Conference on Wireless Communications, Networking and Mobile Computing, WiCom 2007, pp. 2808–2811, September 2007