PTP Security Measures and their Impact on Synchronization Accuracy

Filip Rezabek*, Max Helm†, Tizian Leonhardt‡ and Georg Carle§
I8 Network Architectures and Services, *Technische Universität München* Germany
Email: *rezabek@net.in.tum.de, †helm@net.in.tum.de, ‡leonhart@in.tum.de, §carle@net.in.tum.de

***Abstract*—The Precision Time Protocol (PTP) synchronizes clocks in a network with high precision. The protocol finds use in many areas, such as smart manufacturing, intra-vehicular networks, and critical infrastructure. It becomes clear that striving for security is an important goal. If an attacker succeeds in disturbing the network synchronization, the impact can result in a cascading set of failures. Unfortunately, neither the previous two IEEE standards for PTP, nor the popular implementation `linuxptp`, feature or implement sufficient security options.**

This work focuses on implementing the security extensions for PTP based on the latest PTP standard IEEE 1588-2019 to minimize the threat of attacks and their possible impact. We provide a detailed analysis on PTP synchronicity and security. Based on that, we design and implement software-only tooling to quantify the PTP performance using commercial off-the-shelf hardware and open-source solutions on a linear topology with four to nine hops.

The measurements compare the End-to-End (E2E) and Peer-to-Peer (P2P) delay calculation modes and the usage of Transparent Clocks (TC) in parts of the network. Both E2E and P2P show visible degradation of clock synchronization with each hop and standard deviations of 118.6 to 571 ns. The TCs perform better, demonstrating a standard deviation between 90 to 140 ns on four to nine hops. We evaluate different `logSyncInterval` values corresponding to different PTP profiles and do not observe a major impact on the clock behavior caused by the extensions. The measurement precision of the system is within ±40 ns.

Our evaluation of the newly implemented security extensions to `linuxptp` shows that the security extensions do not have a significant impact on the clock synchronization and our approach is a feasible addition to PTP. Besides, our contributions can aid network managers in assessing their PTP synchronicity systematically.

***Index Terms*—PTP, security, clocks, synchronicity, TSN**

I. Introduction

The Precision Time Protocol (PTP) is standardized in the IEEE 1588 standard [1], [2] and allows for the synchronization of clocks in a network with high accuracies. With hardware timestamping, where timestamps are generated by the Network interface card (NIC) to bypass any non-deterministic delays introduced by the networking stack [2], accuracies in the nanosecond range are possible. PTP is employed in many areas, including power grids [3], [4], finance applications [5], and Time Sensitive Networking (TSN) [6].

The high accuracy requirements result in a high susceptibility to instabilities and deviations in the PTP network, requiring protection mechanisms against attackers deliberately trying to harm the synchronization stability. Successful attacks have the potential to cause catastrophic damages due to PTP being employed in critical infrastructure [4]. On top of that, the topic of security has not been a high priority in the past. The first two PTP standard versions only contain security advice of limited practicality, if any [7]–[9].

To implement security extensions, we use and analyze the popular PTP implementation `linuxptp`, which reveals only basic measures to secure the synchronization stability. As a part of the contributions, we provide an analysis of the security recommendations for PTP described in Annex P of PTPv2.1 [2]. As a result, we determine what extensions should be implemented and what new security guarantees they can provide. The selected solutions ensure message integrity and authenticity and also mitigate replay attacks of PTP messages. A key aspect of deploying PTP into a network is the ability of assessing the precision it can offer. For that, we design a toolchain using open-source solutions and commercial off-the-shelf (COTS) Hardware (HW) that extends the functionality of the EnGINE framework [6]. We devise a set of experiments that assess the baseline performance of PTP for Peer-to-Peer (P2P), End-to-End (E2E), and the use of Transparent Clocks (TC). The default behavior is then compared with scenarios that introduce PTP security extensions to provide insights about their applicability. Besides that, we evaluate `logSyncInterval` values corresponding to various PTP profiles. Network managers can easily reproduce the approach to assess the clock precision in their network. We provide a link to the repository hosting our modified `linuxptp` version and data from individual experiments.

The paper follows a standard structure and the key contributions of the paper can be summarized as follows:

KC1 Analysis of the suggested PTP security extensions and their implementation to `linuxptp`
KC2 Design and development of a measurement toolchain to evaluate PTP clock synchronization
KC3 Evaluation of the impact of security extensions on PTP clock synchronization

II. Background

This section provides fundamental knowledge about PTP and Hash-based Message Authentication Codes (HMAC).

We would like to show our appreciation to Marton Kajo for the discussions during the course of this research paper.

A. PTP

From the three versions of the IEEE 1588 standard, we focus on PTPv2.1, which was released in 2019 and remains compatible with PTPv2. PTPv2.1 is bundled with a new security Annex P [2]. The security measures suggested by Annex P are split to four Prongs (A, B, C, D), each offering distinct proposals.

Prong A focuses on the protocol messages and how to ensure their integrity through symmetric cryptography, as well as authenticating PTP nodes and mitigating the threat of replay attacks. The first proposal also mentions the need for key management to ensure private key protection by providing the necessary parameters. Prong B describes how the existing security options MACSec [10] and IPSec [11] may be employed in a PTP network. Prong C focuses on the resilience of the PTP network by adding redundant Grandmaster Clocks (GMs) and optimizing the topology layout. Last, Prong D proposes how a monitoring solution could be designed and which PTP behavior metrics, such as the link delay and jumps in the master offsets, can help with attack detection.

The PTP standard defines three types of clock devices - Ordinary Clock (OC), Boundary Clock (BC), and TC. The OC is the simplest PTP device. It has one port that can be in either master or slave state. A BC has two or more ports, where all but one are in the master state, and is used to link parts of a PTP topology. The remaining port is in the slave state and is used to synchronize the internal clock of the BC, which is in turn propagated via the master ports. The BC also acts as full-featured PTP node and its synchronized internal clock can be used by applications that need it. A TC, unlike the BC, does not synchronize itself to the time reference. Instead, it forwards the PTP messages and adjusts a time correction field in the PTP message according to the residence time in the TC. Based on [12], using TCs improves the synchronization accuracy over a purely BCs based network.

The GM has a key role in PTP. Any OC or BC may become a GM. The GM is the timing baseline for the PTP network to which other clocks are synchronized. To establish the topology, a GM has to be chosen to dictate the timing baseline. To establish the topology, the GM is configured automatically via the Best Master Clock Algorithm (BMCA), which compares various quality parameters that the clocks self-report.

The IEEE 802.1AS standard, which branches off of IEEE 1588, is specifically tailored to networks for time-sensitive applications [13]. We use the `linuxptp` project for our experiments, which supports various configuration options for this variant and provides various PTP profiles, e.g., the gPTP profile.

1) Message format: For the security extension of `linuxptp`, we need to add new fields that enhance the PTP protocol messages. First, we introduce the message format used in PTP. A PTP packet is prefixed with the *header*, which transports meta-information about the protocol message such as `messageLength`, `messageType` header fields, and the `sequenceID`. Besides, we use different PTP domains, which allows the PTP traffic to be divided into distinct groups and is signified by the `domainNumber` field in the header. The *body* content depends on the type of protocol message to be transmitted. Each message may be suffixed with Type-Length-Value (TLV) appendices. A TLV always begins with the type and length of the data it carries and and is used to transport additional information in protocol messages.

The clock synchronization process relies on PTP nodes exchanging timing information that is used to compute the clock offset and the message path delay between the nodes. Furthermore, one may choose between two distinct delay calculation modes, E2E or P2P.

For both E2E and P2P, four timestamps are collected during the synchronization process in Sync, Follow-up, Delay Req. and Delay Resp. messages. For E2E, the slave calculates the path delay and offset to accurately synchronize itself. In case of the P2P mode, each device observes the delay between its neighbors. The PTP ports on different nodes enter a peer-to-peer relationship and periodically exchange *Pdelay* messages to determine the delay between each other. For both modes, a symmetric delay between master and slave is assumed, otherwise the clock synchronization accuracy suffers [4].

2) `linuxptp`: We use the functionality of `linuxptp`. It comes with multiple binaries, each meant for a different purpose. First, the `ptp4l` program implements a PTP clock and ensures communication among the individual clocks in the networks. Second, the `phc2sys` tool enables the PTP synchronization in applications by synchronizing multiple clocks, such as a physical PTP clock to the system clock. Lastly, `pmc` is used for the PTP management.

3) HMAC: To ensure message integrity and authenticity, we rely on HMAC. It utilizes cryptographic hash functions in a construction that also takes a secret cryptographic key into consideration.[1] Altering the message without possessing the secret key is not possible without it being detectable, ensuring message integrity. A receiver uses an established secret key for message verification. This implies that a message with a valid signature also authenticates the sender as a trusted party.

The main building block is a cryptographically secure hash function, from which the HMAC scheme inherits its strength. Secure hash functions offer collision, pre-image, and second pre-image resistance [15]. Besides being able to re-use any existing hash function, HMAC constructions also keep any added overhead low as not to skew the computational cost of the originally used function [16].

III. Related Work

In this section, we introduce select related work focusing on PTP synchronization accuracy measurements and PTP security extensions.

A. Measuring PTP Synchronization Accuracy

[17] shows six methods for synchronization accuracy measurements of PTP, ranging from hardware approaches to analytical calculations. Each method is studied with a focus

[1]Some hash functions also support a keyed mode without demanding this construction, such as Blake2 [14] presented later.

on feasibility of automated deployment. The authors find the currently available methods insufficient for their goals. Based on the previous insights, two improved variants are presented, making use of data available in the individual PTP nodes.

To aid in the selection of PTP equipment, a systemic testing approach is presented in [3]. Specifically, the authors demonstrate how the synchronization accuracy, as well as the influences of TCs and BCs can be measured for different combinations of PTP HW, mainly COTS. The methodology is targeted at PTP infrastructure maintainers to aid finding suitable equipment fitting their needs.

B. PTP Security Extensions

Another significant part of our contribution is to build security extensions for `linuxptp`. Therefore, we assess various related work that focuses on PTP security.

1) Prong A Security Extensions: [18] follows the Prong A extensions closely, but presents only few implementation details for their work on top of `linuxptp`, e.g., no details on the HMACs used. The results focus on a three-node topology with two OCs connected by a TC. Clock frequency adjustments are measured on the slave clock and compared against measurements with and without a security extension. They also evaluate the increase in residence time for Sync messages and response time for P2P-type messages.

Similarly, [9] extends the `PTPd` implementation by another Prong A security extension. Their evaluation is based on three criteria: feasibility, functionality, and performance. For feasibility, the description of Prong A in the PTPv2.1 standard is assessed for its completeness. Functionality is concerned with the functional aspects of the actual implementation. Lastly, the performance is evaluated with regard to the increase in message residence time, as shown in [18].

Another approach and solution to the Prong A extension is presented in [8]. It is based on a high-level description of the four Prongs in an early draft of Annex P, during which was not clear separation of individual Prongs. The authors also feature an in-depth analysis of PTP security, which is a good supplementation for our security analysis. The performance of the implementation for `PTPd` is evaluated based on the processing overhead introduced by the cryptographic measures.

2) Other Security Extensions: The detection mechanisms presented in [4], [19], [20] take a more passive approach to security. The authors propose the addition of watchdog nodes that observe the PTP traffic behavior and detect abnormalities. Once detected, the system can fall back to a secure state where an attacker has less influence until the incident is handled. [21] focuses on the management of PTP keys, which is not covered within the standard itself. We also want to point out the survey paper [22], providing an overview of other PTP related work.

IV. Analysis & Design

In comparison to other related work, we analyze the security extensions of Prong A in more detail, and scale the experiments over more complex topologies using the newly designed toolchain. Based on the analysis of the extensions for PTP, we design and implement the selected extensions for `linuxptp` to assess the impact on the overall system synchronicity.

A. PTP Synchronicity

To evaluate the impact of the security extensions on synchronization accuracy, we need to design a suitable toolchain. The toolchain allows an assessment of the PTP baseline performance and its comparison to deployments with security extensions. There is no single definition for the synchronization precision requirement of a PTP system [2], as it highly depends on the deployments and their corresponding PTP profiles. In applications like smart power grids, a synchronicity of at least $1\,\mu s$ [4] is required.

The toolchain should allow for the evaluation of requirements for a given use-case. For that, we need to quantify the concept of synchronicity, enabling us to satisfy the demands and that the extensions we build are a viable addition to `linuxptp`. Multiple approaches exist to determine the synchronicity, such as using an oscilloscope [17], simulation [17], or in software with real PTP networks.

We build our solution using COTS HW that supports 802.1AS and open-source solutions for measurements in software. We integrate the measurement setup in the EnGINE framework [6], which allows for evaluation at scale, reproducible experiment execution, and a high level of automation.

B. PTP Security

Next, based on RFC 7384, we define security requirements for PTP. Each of the requirements has a different level of priority for PTP [23]. Table I presents the security requirements, grouped based on their relation to requirements and attack protections [23]. Besides, it shows what attacks these requirements mitigate from an experimental perspective to understand what advantages it brings to the network managers. Note that we omit MAY and most SHOULD requirements.

TABLE I: Security requirements classification [23]. ”-” means that the requirement does not directly correlate with an attack

Req. # and Requirement	Protects against
MUST	
R1: Authentication & authorization	Spoofing, Rogue master attack
R2: Integrity protection	Manipulation
R3: Spoofing prevention	Spoofing
R4: Replay protection	Replay attack
R5: Key freshness	Replay attack
R6: No performance degradation	-
R7: Delay and interception protection	Interception and removal, Packet delay manipulation
R8: Secure Mode	-
SHOULD	
R9: Time protocol DoS protection	Time protocol DoS attacks

As outlined, we focus on Prong A, which enhances the PTP stack by employing symmetric cryptography to ensure message integrity and authenticity along with replay attack protection. Therefore, we discuss the design decisions to be made before extending PTP with HMAC and additional implementation aspects.

Prong A defines the so-called `AUTHENTICATION` TLV, which may be attached to any message that is to be protected. It contains the HMAC value of the given message content. The Prong requires the usage of two security databases that store and distribute the parameters for secure message handling.

The PTP network can work in two distinct modes of message processing - immediate and delayed processing. First, the immediate processing of PTP messages checks for a valid `AUTHENTICATION` TLV upon message arrival. If the check fails, the message is dropped. This requires all security parameters to be known up-front.

Second, the delayed processing mode allows for delayed verification of PTP messages. It is beneficial to scenarios where not all cryptographic parameters are known at message reception. In this work, we implement the immediate processing, for which a fast HMAC with low computational overhead is crucial. Such an approach also ensures satisfying **R6**. The various PTP profiles, among other parameters, define how often the synchronization procedure happens based on the `logSyncInterval` option. For the gPTP profile, the synchronization interval is set to $0.125\,\mathrm{s}$ [24]. The overhead should not interfere with the performance of even lower intervals.

The performance can be affected by larger message sizes and processing delays. When comparing HMACs, we compare the message sizes using packet captures of `linuxptp`. The results show message sizes ranging from 44 to $64\,\mathrm{B}$, depending on the profile and settings used, and can possibly be greater with additional TLVs attached.

C. HMAC Selection

We select three HMACs algorithms for further evaluation, as each of them shows a different base performance. Using the baseline performance results, we can evaluate the sensitivity of our measurement setup. As there are many more HMAC algorithms that behave differently, we introduce dummy delays which stall message processing by $100\,\mu\mathrm{s}$. Besides, it provides an understanding of the performance impact of the computational overhead on the individual nodes.

1) *HMAC-SHA-512-256* is based on the SHA-512 algorithm, yields hashes that are truncated from 512 to $256\,\mathrm{bit}$, and takes approximately 13.1 CPU cycles per a $1024\,\mathrm{B}$ message [25]. It is widely-used and implemented in the `libsodium` [26] cryptographic library.

2) *Blake2* is a hash that offers two variants, namely Blake2b and Blake2s, optimized for 64 or below 64 bit architectures, respectively [27]. We use the Blake2b variant. This hashing method promises to offer significantly better performance than the HMAC-SHA-512-256 [27].

3) *Blake3* improves on its predecessor Blake2 by offering a measurable increase in performance. For a message size of $128\,\mathrm{B}$, hashing with Blake3 uses approximately 3.1 CPU cycles per byte (Blake2b: approximately 3.6) [28]. For larger message sizes, the performance advantage of Blake3 increases significantly.

D. Security Databases & AUTHENTICATION TLV

Prong A requires the usage of the Security Policy Database (SPD) and Security Association Database (SAD), which store and distribute the parameters surrounding the secure message content. The standard only dictates the general PTP client interaction with the databases.

The SPD contains the information needed to evaluate if a message is to be secured. It is queried by passing so-called policy-limiting fields, i.e., parameters that allow narrowing down the selection as much as possible. All fields are part of the PTP message header and allow for fine-grained policy control. The actual policy values stay static over the lifetime of the PTP instance. Part of the results are the Security Parameter Pointers (SPPs) that point into the SAD.

The SAD is responsible for storing information about Security Associations (SA), i.e., the set of parameters that are needed to perform the securing of messages. Using the SPPs, the SAD can be queried to return the corresponding SA. This allows easy sharing of the SPP value, as the instances establish the same association without having to send cryptographic secrets in-band. The SAD also keeps track of the `sequenceID` field of the header in received PTP messages.

Contrary to the SPD, the contents of the SAD *may* change during the runtime of a PTP instance. This is useful when the PTP application needs to update the cryptographic keys, which is needed for **R5**. Changing cryptographic parameters during the runtime requires automated key management to provide the initial secrets and refresh them. Just like the actual implementation of SAD and SPD, the key management itself is not in the scope of the PTPv2.1 standard. We rely on manual key management for our needs, as the main goal is to assess the performance impact of security extensions. In production deployments, it is crucial to follow recommendations for proper key management as mentioned in [21].

The SAs establish the parameters required for the security processing. This includes a symmetric key for keyed hashing, as well as the corresponding algorithm to be used in the integrity-check calculation.

The `AUTHENTICATION` TLV is attached to any message that requires protection based on the SPD information. It is usually placed as the last TLV in a PTP message. Figure 1 shows the individual parts of the `AUTHENTICATION` TLV, along with the field sizes in bytes. The size of the `disclosedKey` and `ICV` fields depends on the hashing algorithm used. The `sequenceNo` and `RES` fields currently have a size of zero and are reserved fields [2].

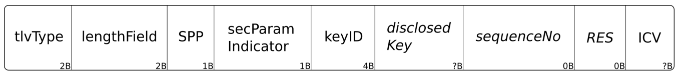


Fig. 1: The structure of the `AUTHENTICATION` TLV [2]. Italicized text indicates an optional field.

The TLV begins with the type and length of the following content. This is continued with the SPP that points inside the SAD and associates the message with the corresponding set of security parameters. The presence of the three optional fields (shown in italicized writing) is indicated by the `secParamIndicator` bit-field. The `keyID` is used to uniquely identify a key. Only `disclosedKey` is an optional field used in the delayed processing mode. The last part of the `AUTHENTICATION` TLV is the Integrity Check Value (ICV), which contains the hash of the message. The hash is calculated over the PTP header, body, and any TLV fields before it and ensures the integrity and authenticity of the message.

Additional details are provided in the enclosed repository[2].

E. Summary of Security Gains

With the `AUTHENTICATION` TLV, we achieve integrity and authenticity of messages using symmetric cryptography. By keeping track of the `sequenceID` in protocol messages, we mitigate replay message attacks. Therefore, based on Table I, we fulfill **R2**, **R3**, and **R4**. Since **R9** is achieved by introducing a mechanism for authentication [23], we consider it fulfilled on the time protocol layer.

Section VI shows that **R6** is also fulfilled. Since the SPD may be configured to demand authentication for every protocol message, using **R8** is possible, and the respective requirement is satisfied.

V. Experiment Setup

This section details the architecture of our toolchain and measurement methodology.

A. Measurement Setup

To achieve reproducible and scalable measurements, we utilize the EnGINE framework [6]. The experiments are conducted in four steps, enabling testing of various network topologies and PTP configurations. Of note, the installation of nodes relies on the automated setup via the *pos* framework [29].

Our toolchain is run on up to 13 physical nodes, wired as shown in Figure 2. The nodes are built using COTS HW and NICs that support additional TSN standards, such as Intel `i210`, `i350`, `x552`, and Cisco Nexus GM. These NICs support hardware timestamping, ensuring a good synchronization baseline that is crucial for our experiments. Each NIC has multiple Physical Hardware Clocks (PHCs) as available interfaces, with the exception of the Cisco Nexus GM which has only a single PHC. Hence, we need `phc2sys` to synchronize the individual PHCs. The Cisco Nexus GM NIC can be synchronized via GPS and offers a timestamping resolution of 4 ns [30], present at `lisa`. The node `maggie` serves as our GM to provide the time reference.

B. Data Collection & Methodology

To quantify the PTP system synchronicity, we compare a reference time value to one that results from the behavior of the usual protocol flow in a realistic topology. To achieve this, we start with a direct connection from `maggie` to the node we do our measurements on, which is assumed to be `abe` from now on (cf. Figure 2, the red square box shows the setup).

[2]https://cnsm-ptp-security.pages.gitlab.lrz.de/ptp-security/

The time of the PHC synchronized via the slave port on `abe` is our reference value. Since there are no additional hops in between, we assume that this is sufficiently precise. To realize the second time value, another port on `abe` has to be synchronized with `maggie` serving as the GM once again to ensure having the same timing baseline. This second port is synchronized via a different path that we can design according to our needs. We increase the number of hops on the *long path* between `maggie` to `abe`. We use a linear topology, as it is often used in real-world PTP applications, such as in industrial automation [17], [31], and allows for better performance assessment over more hops.

Figure 2 shows an example of two and four hop setups, where we extend the topology by adding additional nodes. To run two `linuxptp` instances on the same node, we need to split different paths to different domains. The short path is colored red and runs in PTP domain 1. We also show the long path, which runs in PTP domain 0 and is colored green or blue for two or four hops, respectively. Besides, for the case of P2P and E2E, we show which devices are selected as master and slave interfaces. For the experiments evaluating the impact of security extensions with TCs on the hops, we highlighted the nodes with a red dot in a given example. Overall, any node on the long path runs as a TC, except for the GM `maggie` and `abe`, which is used for the PHCs value comparison. Such an approach enables us to build a realistic PTP topology on the long path and scale the number of nodes arbitrarily until reaching the maximum number of hops available in the testbed. We believe that evaluating the impact of fewer hops can be extended to a model for a larger number of hops.

To ensure the comparison of the time values happens at the same point in time, we rely on the `phc_ctl` tool that is bundled with `linuxptp`. It offers a comparison operation to check the system clock offset relative to a PHC in the form of `ioctl` system calls. Hence, we synchronize one of the PHCs to `CLOCK_REALTIME` with `phc2sys` and compare `CLOCK_REALTIME` to the second PHC using `phc_ctl`. In Figure 2, the PHC of the port on `abe` that is active in domain 1 is used to synchronize `CLOCK_REALTIME`, which is compared with the PHC on the long path in domain 0.

Since `phc2sys` uses the same `ioctl` as the `phc_ctl` for the system clock synchronization and is built into `linuxptp` [24], we deem the accuracy sufficient to evaluate the impact of security extensions on the performance.

1) Methodology: The measurements focus on evaluating the impact of the security extensions on clock synchronization over a varying number of hops on the long path. We measure four to nine hops, and for each number of hops we evaluate the baseline (no security extension) and the hashing methods for the ICV. Besides, we introduce dummy HMACs that sleep for 100 µs with a random factor between ±30 µs during ICV generation and verification. This helps to model the overhead caused by security extensions that rely on more expensive cryptographic operations. Table II shows the pool of hashing methods we use, alongside the most important parameters.

For the description of the results, we focus on the absolute

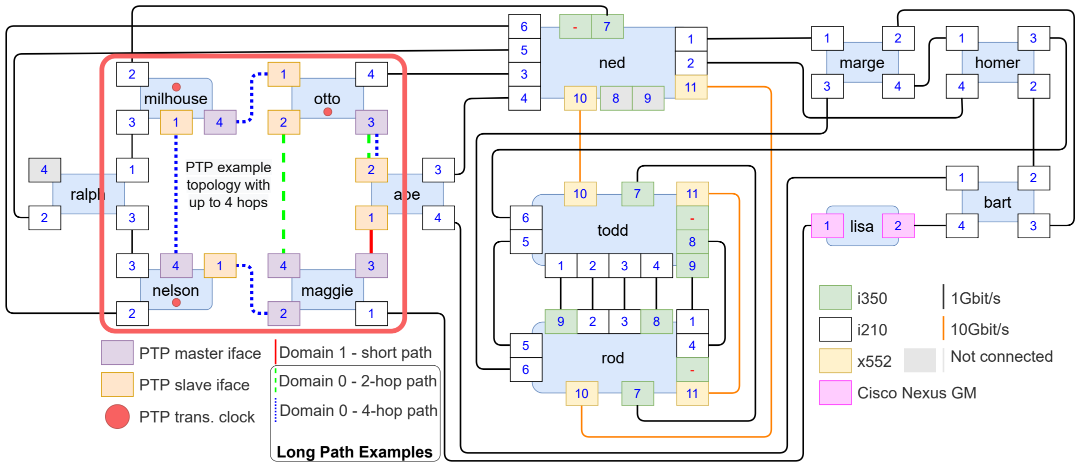


Fig. 2: Overview of the Springfield testbed and example topology with up to 4 hops and marking of TC and OC

clock offset from the correct time (i.e., the mean sync offset) and the standard deviation σ of the offset values. The absolute offset shows the quality of the synchronization mechanism and σ describes the variation of the offset values. We can show how stable the achievable synchronization is using σ. A larger value of σ indicates a less stable clock synchronicity.

TABLE II: Hashing methods considered for our result measurements. Key size (k) and output size (o) in bytes

Algorithm	Parameters
HMAC-SHA-512-256	`k=32 o=32`
Blake2b	`k=32 o=32`
Blake3	`k=32 o=32`
Dummy	Sleeps 100 µs, random factor ±30 %

Our configuration for `linuxptp` relies on the link layer message transport and a `logSyncInterval` of -7, i.e., initiating clock synchronization every 2^{-7} s (≈8 ms). We empirically determined this value to be both the most stable and most accurate interval on the testbed, starting with a value of 2^{-12} until we found the final interval with a standard binary search approach within the duration of a single experiment. To match the synchronization period of PTP, we measure the clock every 8 µs using the `phc_ctl` tool for a total of 450 000 data points, i.e., measuring roughly one hour for each experiment. Due to a bug, not all data points are used for evaluation. The reason and solution to this challenge are described in Section VI-B.

All nodes in the testbed run Ubuntu 20.04 LTS with Linux kernel version 5.4 and use only the Intel `i210` NIC. Cisco Nexus GM is used for the setup evaluation.

VI. Results

The results present the evaluation of our setup precision, where we compare clock offsets on `lisa` and `abe`, and motivate why the offset is not the only relevant metric to look at and why the standard deviation σ provides additional insights. Next, we showcase two additional experiments comparing the E2E and P2P modes, as well as using TCs on the hops. For the last two experiment campaigns, we run them over four to nine hops each and compare a baseline (no security extensions) with three different HMACs, and a dummy value. Next, we provide details on the residence time collected on a single node for all of the setups (no-/security, dummy) using TCs. Finally, we assess the impact of various `logSyncInterval` values on the precision with BCs and TCs on 9 hops with and without security extensions. The experiments answer whether our security measures significantly impact the synchronization accuracy negatively and if our measurement setup allows measuring at a fine enough resolution to detect the newly introduced overhead. To note, we use the two-step mode.

A. Setup Evaluation

To evaluate the precision of our setup, which relies on software, the fundamental difficulty is the comparison of two time values at the same instant. To understand the error this approach introduces, we run an additional experiment where we synchronize a PHC to `CLOCK_REALTIME` and then compare it to the *same* PHC via `phc_ctl`. This highlights two error sources—the synchronization of a PHC to another clock and the approximate comparison operation. They represent the main bottlenecks for the accuracy of our results. We run the experiment on `lisa` and `abe` to assess the fluctuations.

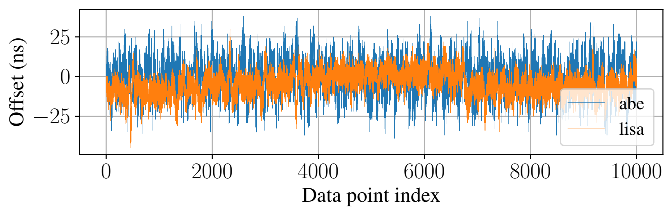


Fig. 3: Offset fluctuations for `abe` (blue) and `lisa` (orange)

Based on Figure 3, we see offset fluctuations of ±40 ns for `abe` and ±30 ns for `lisa`. Since we compare the clock values of the two domains on `abe`, the error we deduce is predictable but could skew the results, making the comparison on a small scale difficult. We decided to use the Intel i210 NIC as GM, as it is a more affordable and commonly used NIC, ensuring

TABLE III: The mean for P2P and E2E in ns for with and without security extensions over different number of hops

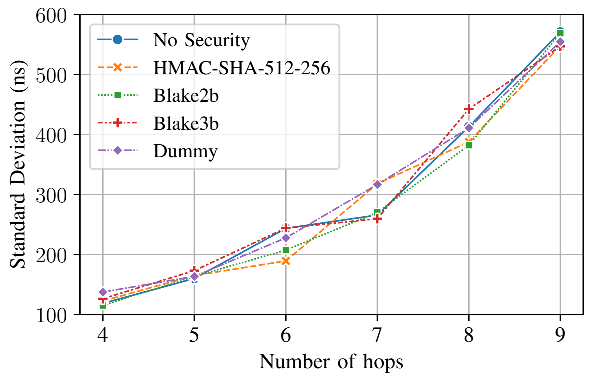
Security	4 Hops $\bar{x}$	5 Hops $\bar{x}$	6 Hops $\bar{x}$	7 Hops $\bar{x}$	8 Hops $\bar{x}$	9 Hops $\bar{x}$
P2P						
No Security	2.2	7.7	-4.6	-10.5	-19.1	3.6
HMAC-SHA-512-256	8.0	2.6	3.8	-14.9	-26.5	-41.9
Blake2b	1.6	16.0	12.7	-10.7	-4.5	-53.3
Blake3b	23.8	8.1	13.5	-6.3	-16.4	-43.3
Dummy	14.5	10.3	10.7	-20.3	-21.5	-55.0
E2E						
No Security	-14.3	-15.3	-26.2	-29.3	-38.4	-58.5
HMAC-SHA-512-256	-8.9	-17.2	-17.1	-43.1	-57.2	-74.3
Blake2b	-13.9	-17.6	3.3	-21.4	-47.6	-21.3
Blake3b	-24.1	-15.3	-17.2	-23.8	-27.9	-23.6
Dummy	-12.6	-19.8	-10.7	-27.2	-38.3	-23.4

use of homogeneous NICs in the experiments. This experiment shows the possible advantages of a more precise GM, such as the Cisco Nexus, in comparison to COTS HW solutions.

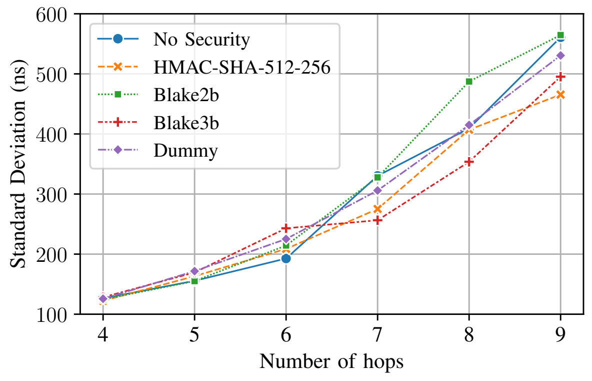
1) Comparison of P2P and E2E: Table III shows mean values for P2P and E2E, with all values having a negative bias. Unfortunately, the results do not provide sufficient information to assess the impact of security extensions. Therefore, Figure 4 shows the standard deviation σ. The σ values provide important observations regarding the PTP synchronization. It confirms our assumption that the standard deviation describes the variation of the offset values, providing insights into the clock synchronization stability. As expected, σ increases with more hops and with higher delay. For P2P, Figure 4a shows that the synchronization offset increases significantly with each hop introduced on the long path. The standard deviation for four hops lies between 118.6 ns and 137.4 ns, increasing to a range between 546.9 ns and 571 ns for nine hops. We can see that the synchronization offset deteriorates over increasingly long linear paths, which matches our expectations and the insights for long linear PTP topologies.

The standard deviations for each security option are similar within a hop group. For four hops, all values stay within 18.8 ns of each other, increasing up to a range of 24.1 ns for nine hops. A similar tendency is also visible for E2E mode, shown in Figure 4b. There is no definitive correlation between the computational overhead introduced by our security measures and the standard deviation. This is evidenced by the fact that for all hop groups (excluding six hops) shown in Figure 4, there is always at least one measurement that has a lower standard deviation than the case without any security measures in place. The fluctuation in standard deviation is visible for all other security options used. Based on these facts, we can observe no increase in deviation of the synchronization offset values within the same hop configuration that we can measure. This might be due to the precision being below the fluctuations. Nevertheless, since the impact on a per-hop basis is visible, we believe our security extensions do not cause a significant impact, i.e., less than the per-hop impact.

2) Performance with TCs: Similar to the previous experiments, we choose E2E for the experiments with TCs, as there



(a) P2P



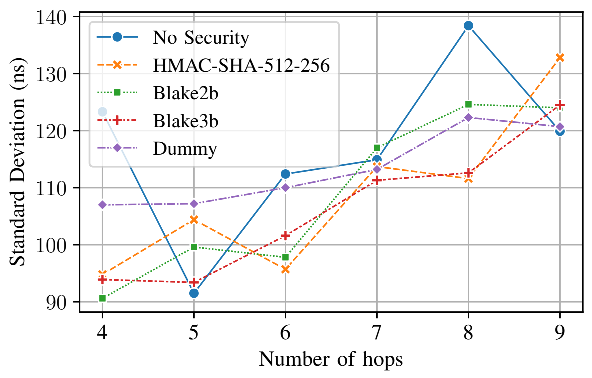
(b) E2E

Fig. 4: The standard deviation in ns for P2P and E2E

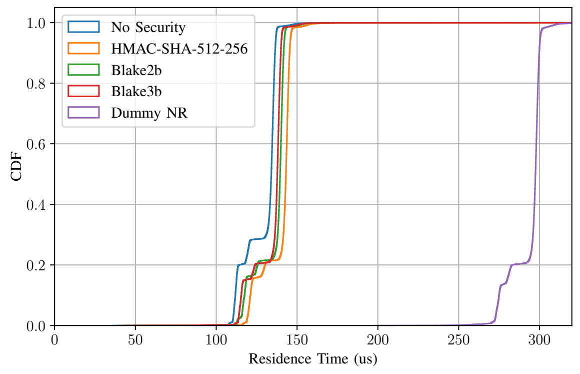
is no significant performance difference between P2P and E2E. In this case, `maggie` and `abe` run in E2E mode and the rest of the hops on the *long path* are configured with TCs.

Figure 5a shows the standard deviation over 4-9 hops and all paramaters used. In comparison to P2P and E2E, we see less increase with each hop. For four hops the standard deviation lies between 90.6 ns and 123.3 ns, and for 9 hops it ranges between 119.9 ns and 132.8 ns. We can see that the behavior of different operation modes also does not show a clear tendency and all options behave similarly. For four hops, values are within 32.7 ns of each other and within a range of 13 ns for nine hops. We assume that this is due to minor differences and the fluctuations are within the error bounds and precision which our setup can identify.

Finally, the residence time within a TC on the individual hops is also relevant. This value is added to the correction field to ensure a correct time value being forwarded, unaffected by the processing done by TCs. Figure 5b shows the residence values for all setups. No security yields the lowest residence times compared with security being in place, but the difference is within a few µs. The minuscule overhead compared to the no security case affirms the viability of our security extension. The dummy result shows the expected impact, adding 150 µs to the usual residence time. The additional overhead of 50 µs is caused by the `nanosleep` function, resulting in a total residence time of approximately 300 µs.



(a) The standard deviation in ns for TC



(b) The CDF of the residence time for a TC hop in µs

Fig. 5: TCs σ per hop and residence times on a hop

3) Performance of `logSyncInterval`: To assess various PTP profiles, we evaluate `logSyncInterval` values from -7 to 0 for the BCs and TCs setups. As identified previously, E2E and P2P does not present a significant difference, so we focus on E2E. Both setups are assessed over over 9 hops, with and without security extensions.

Even though none of the profiles requires such a low `logSyncInterval` value, we still decided to use -7 to assess the performance in scenarios where the possible overhead caused by the additional processing overhead could be significant. As mentioned in the Section V, the value of -7 is determined empirically. To note, there are additional requirements for the profiles, such as using unicast instead of multicast, which are outside of the scope of this paper.

Figure 6 shows the results for BCs and TCs setups. As expected, for the BCs we see a more significant impact on the synchronization accuracy with varying values of `logSyncInterval`, ranging from 546.9 to more than 3200 ns for a value of 0. On the other hand, for the TCs, the difference is within the system accuracy error with values of 120±40 ns. As outlined in previous results, we cannot observe the additional overhead caused by the security extensions or the dummy values.

Overall, the various PTP profiles relying on BCs have to consider the possible decrease of system accuracy with higher values of the `logSyncInterval`, but do not have to compromise on security goals.

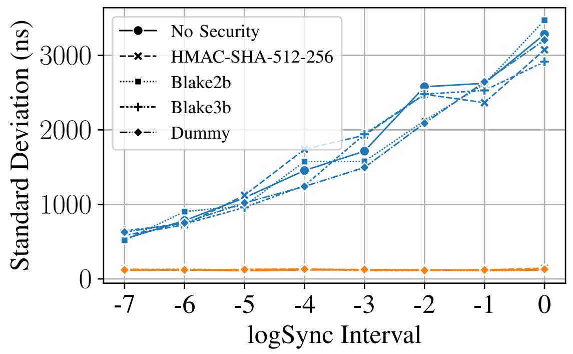


Fig. 6: Standard deviation in ns for BC and TC with various `logSyncInteval` values

B. Setup Challenges & Limitations

1) Setup challenges: We encountered two main challenges with our measurement setup. During the measurements, we encountered a visible error during the data sampling. It can be observed whenever the synchronization with the GM is lost, causing the clocks to deviate heavily and taking several seconds to get in sync. This is most likely caused by a bug in the Linux kernel, but additional investigation is needed.

Since we did not find a way to prevent this, we decided to cut the affected intervals. We search for 1200 values that are continuously beyond 1.5 times the Inter-quartile Range (IQR), which reliably identifies the bug. We chose 1200, as it roughly corresponds to a 10 s interval. Within this time, synchronicity is again achieved within the network. Once found, we cut 1200 uninterrupted data points and remove 1000 data points from either side of the interval to clean the data before and after the destabilization. This approach allows for reliable detection of the wrong data without removing normal behavior data.

2) Security extensions: The selected security extensions do not fulfill all of the MUST and SHOULD requirements from Table I. For instance, to satisfy **R7**, one may engage the mechanisms of Prong C and Prong D or choose an external solution like [4] or [20].

R1 is met partially, since we provide a mechanism for authentication, but not authorization. The nodes can verify other nodes' authenticity, but can not ascertain whether they act within their permitted bounds. **R5** is not met due to the manual key management we employ. This is not problematic, as the implementation can be expanded to enable it, and proper key refresh policies should be used in production deployments.

Nevertheless, all of these requirements are outside of the scope as we focused on extensions that can be directly integrated into the PTP protocol and do not require external entries and policies to be fulfilled.

VII. Conclusion & Future Work

This work shows that adding security extensions to PTP in the `linuxptp` project does not have a significant impact on the clock synchronization. We have designed and evaluated a PTP network using COTS HW and a software-only solution.

The designed methodology and toolchain enabled us to assess the performance for different modes—P2P, E2E, and using TCs on the hops with and without security extensions. Using TCs on the hops yields better results than using P2P and E2E on the hops. When evaluating the `logSyncInterval` values for different PTP profiles, we did not observe an additional impact on clock deviation caused by the security extensions. However, the `logSyncInterval` has an impact on the system performance following the trends of the BCs and TCs experiments. The error caused by the system is within $\pm 40\,\mathrm{ns}$, which is within the variance of experimental results on a given hop. In addition, to evaluate what time of computational overhead on each hop is feasible, we designed experiments with dummy values that introduce an additional delay. This systematic approach can be applied in many networks and can help with the administration of PTP networks. The extended codebase, experiment data, and their processing is available on GitHub, with additional implementation details.

In future work, we plan to improve the accuracy of the system by possibly adding new COTS HW supporting the cross-timestaming kernel feature and lower the operating system overhead by CPU affinity and isolation. Next, `linuxptp` can be extended with additional security extensions, e.g., public key cryptography or additional symmetric key algorithms and incorporate solutions for key management. Furthermore, we could assess various profile requirements in more details. The described measurement toolchain can be used to evaluate the impact on the performance of such solutions and configurations.

References

[1] "IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems," July 2008.

[2] "IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems," June 2020.

[3] D. M. Ingram, P. Schaub, D. A. Campbell, and R. R. Taylor, "Performance Analysis of PTP Components for IEC 61850 Process Bus Applications," *IEEE Transactions on Instrumentation and Measurement*, vol. 62, no. 4, pp. 710–719, 2013.

[4] B. Moussa, M. Debbabi, and C. Assi, "A Detection and Mitigation Model for PTP Delay Attack in a Smart Grid Substation," in *2015 IEEE International Conference on Smart Grid Communications (SmartGridComm)*, IEEE, Nov. 2015.

[5] P. V. Estrela and L. Bonebakker, "Challenges Deploying PTPv2 in a Global Financial Company," in *2012 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control and Communication Proceedings*, IEEE, Sep 2012.

[6] F. Rezabek, M. Bosk, T. Paul, K. Holzinger, S. Gallenmüller, A. Gonzalez, A. Kane, F. Fons, Z. Haigang, G. Carle, and J. Ott, "EnGINE: Developing a Flexible Research Infrastructure for Reliable and Scalable Intra-Vehicular TSN Networks," in *3rd International Workshop on High-Precision, Predictable, and Low-Latency Networking (HiPNet 2021)*, (Izmir, Turkey), Oct. 2021.

[7] C. DeCusatis, R. M. Lynch, W. Kluge, J. Houston, P. A. Wojciak, and S. Guendert, "Impact of Cyberattacks on Precision Time Protocol," *IEEE Transactions on Instrumentation and Measurement*, vol. 69, pp. 2172–2181, May 2020.

[8] E. Itkin and A. Wool, "A Security Analysis and Revised Security Extension for the Precision Time Protocol," *IEEE Transactions on Dependable and Secure Computing*, vol. 17, pp. 22–34, Jan. 2020.

[9] D. Maftei, R. Bartos, B. Noseworthy, and T. Carlin, "Implementing Proposed IEEE 1588 Integrated Security Mechanism," in *2018 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control, and Communication (ISPCS)*, IEEE, Sept. 2018.

[10] "IEEE Standard for Local and Metropolitan Area Networks-Media Access Control (MAC) Security," 2018.

[11] R. Atkinson, "Security architecture for the Internet Protocol." RFC 1825, Aug. 1995.

[12] D. Fontanelli and D. Macii, "Accurate Time Synchronization in PTP-Based Industrial Networks with Long Linear Paths," in *2010 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control and Communication*, pp. 97–102, 2010.

[13] "IEEE Standard for Local and Metropolitan Area Networks–Timing and Synchronization for Time-Sensitive Applications," 2020.

[14] M.-J. O. Saarinen and J.-P. Aumasson, "The BLAKE2 Cryptographic Hash and Message Authentication Code (MAC)." RFC 7693, Nov. 2015.

[15] S. Al-Kuwari, J. H. Davenport, and R. J. Bradford, "Cryptographic Hash Functions: Recent Design Trends and Security Notions," *Cryptology ePrint Archive*, Jan. 2011.

[16] D. H. Krawczyk, M. Bellare, and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication." RFC 2104, Feb. 1997.

[17] S. Schriegel and L. Wisniewski, "Investigation in Automatic Determination of Time Synchronization Accuracy of PTP Networks with the Objective of Plug-and-Work," in *2014 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control, and Communication (ISPCS)*, IEEE, Sept. 2014.

[18] E. Shereen, F. Bitard, G. Dan, T. Sel, and S. Fries, "Next Steps in Security for Time Synchronization: Experiences from Implementing IEEE 1588 v2.1," in *2019 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control, and Communication (ISPCS)*, IEEE, Sept. 2019.

[19] E. Itkin and A. Wool, "A Security Analysis and Revised Security Extension for the Precision Time Protocol," *IEEE Transactions on Dependable and Secure Computing*, vol. 17, no. 1, pp. 22–34, 2020.

[20] W. Alghamd and M. Schukat, "A Detection Model Against Precision Time Protocol Attacks," in *2020 3rd International Conference on Computer Applications & Information Security (ICCAIS)*, IEEE, 2020.

[21] M. Langer, S. Fries, M. Rohde, K. Heine, D. Sibold, and R. Bermbach, "PTP Security Key Management Solutions," in *2021 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control, and Communication (ISPCS)*, pp. 1–6, 2021.

[22] N. Moreira, J. Lazaro, J. Jimenez, M. Idirin, and A. Astarloa, "Security Mechanisms to Protect IEEE 1588 Synchronization: State of the Art and Trends," in *2015 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control, and Communication (ISPCS)*, IEEE, Oct. 2015.

[23] T. Mizrahi, "Security Requirements of Time Protocols in Packet Switched Networks," RFC 7384, Oct. 2014. Accessed on 2022-03-13.

[24] R. Cochran, "linuxptp." Accessed on 2022-01-22.

[25] S. Gueron, S. Johnson, and J. Walker, "SHA-512/256," in *2011 Eighth International Conference on Information Technology: New Generations*, pp. 354–358, 2011.

[26] F. Denis, "libsodium," June 2013. Accessed on 2022-03-05.

[27] J.-P. Aumasson, S. Neves, Z. Wilcox-O'Hearn, and C. Winnerlein, "BLAKE2: Simpler, Smaller, Fast as MD5," in *Applied Cryptography and Network Security* (M. Jacobson, M. Locasto, P. Mohassel, and R. Safavi-Naini, eds.), (Berlin, Heidelberg), pp. 119–135, Springer Berlin Heidelberg, 2013.

[28] J. O'Connor, J.-P. Aumasson, S. Neves, and Z. Wilcox-O'Hearn, "BLAKE3: One Function, Fast Anywhere." Accessed on 2022-03-10.

[29] S. Gallenmüller, D. Scholz, H. Stubbe, and G. Carle, "The pos Framework," in *Proceedings of the 17th International Conference on emerging Networking EXperiments and Technologies*, ACM, Dec. 2021.

[30] "Cisco Nexus GM Time Synchronization NIC Data Sheet." Accessed on 2022-03-12.

[31] M. Schumacher, L. Wisniewski, J. Jasperneite, and S. Schriegel, "Node to Node Synchronization Accuracy Requirements of Dynamic Frame Packing," in *2013 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control and Communication (ISPCS) Proceedings*, pp. 53–58, 2013.