DIPLOMA THESIS

Secure Stack for Clocks Synchronized by IEEE 1588

Submitted at the Faculty of Electrical Engineering and Information Technology, Vienna University of Technology
in partial fulfillment of the requirements for the degree of
Master of Sciences (Diplomingenieur)

under supervision of

O. Univ. Prof. Dipl.-Ing. Dr.techn. Dietmar Dietrich,
Ass.-Prof. Dipl.-Ing. Dr.techn. Thilo Sauter,
and Dipl.-Ing. Albert Treytl

by

Bernd Axel Hirschler, BSc
Matr.Nr. 0026521
Bergweg 352, 2732 Würflach

Würflach, March 2011 _____

Kurzfassung

Diese Diplomarbeit analysiert die Auswirkungen von Sicherheitsprotokollen auf das IEEE 1588v2 Uhrensynchronisationsprotokoll. Der Schwerpunkt liegt dabei auf der hochgenauen Uhrensynchronisation. Im Zuge dieser Arbeit werden die Mechanismen, die zur Sicherung der Netzwerkpakete genutzt werden, analysiert.
Die Analyse von Annex K des IEEE 1588v2 Standards fand einen fundamentalen Fehler, der das Precision Time Protocol (PTP) anfällig für Man-in-the-Middle-Angriffe macht und der es ebenfalls ermöglicht einen Replay-Angriff durchzuführen.
Die Sicherheitserweiterung vom Precision Time Protocol (PTP) wird weiters mit dem Internet-Protocol-Security (IPsec-) Protokoll, einem weit verbreiteten Sicherheitsprotokoll das eine gesicherte Kommunikation basierend auf dem Internet Protocol (IP) ermöglicht. Diese beiden Protokolle werden herangezogen, um die Faktoren zu identifizieren, die die Uhrensynchronisation beinflußen. Die durchgeführten Messungen zeigen, daß die verschiedenen Sicherheitsprotokolle einen deutlichen Einfluß auf die Packet Delay Variation (PDV) haben. Bei weiteren Messungen wird gezeigt, dass die Packet Delay Variation, die durch die Sicherheitsprotokolle hervorgerufen wird, jedoch nur geringe Auswirkungen auf die Qualität der hochgenauen auf Software basierenden Uhrensynchronisation hat.

Abstract

This diploma thesis analyzes the impact of security protocols on clocks synchronized by IEEE 1588 v2. An emphasis is put on high-precision clock synchronziation. The work investigates the security mechanism used to protect the traffic, which carries timing information.
An security analyzis performed on Annex K of IEEE 1588v2, the native solution, reveals a fundamental flaw in the protocol, which makes the Precision Time Protocol (PTP) network prone to man-in-the-middle and replay attacks.
The security extension of the Precision Time Protocol (PTP) is compared against the Internet Protocol security (IPsec), which is a widespread protocol used to establish secure communication for Internet Protocol (IP) packets. These two security schemes are used to explore the influencing factors, which affect the precision of the clock synchronization. The performed measurements show that the various security protocols have a clear impact on the Packet Delay Variation (PDV). Further measurements show that the Packet Delay Variation introduced by the security protocols has only a minor impact on the quality of the high-preccision clock synchronization based on software implementations

Table of Contents

1 Introduction **1**
- 1.1 Motivation . . . 2
- 1.2 Goal of this Work . . . 2

2 Technology Overview **5**
- 2.1 Application of Clock Synchronization . . . 5
- 2.2 Protocol Overview . . . 7
  - 2.2.1 Network Time Protocol – NTP . . . 7
  - 2.2.2 Precision Time Protocol – IEEE 1588v2 . . . 12
- 2.3 Security . . . 17
  - 2.3.1 Network Time Protocol-Security Extensions . . . 19
  - 2.3.2 Precision Time Protocol-Security Extension . . . 21
  - 2.3.3 Internet Protocol Security . . . 26

3 Security Analysis for Clock Synchronization **33**
- 3.1 Attack on Networks . . . 33
- 3.2 Inefficiency at the Startup of Secure PTP Networks . . . 35
- 3.3 PTP Security Flaw . . . 36
  - 3.3.1 Modification of Source Address . . . 36
  - 3.3.2 Creating Counterfeit Security Association . . . 37
  - 3.3.3 Resetting Security Association by Timeout . . . 39
  - 3.3.4 Replay Attack with Counterfeit Security Association . . . 39
  - 3.3.5 Replay Attack with Static Security Association . . . 40
  - 3.3.6 Effect of the Vulnerability . . . 40

4 Design and Implementation of a Secure Clock Synchronization Stack **43**
- 4.1 Design Goals . . . 43
- 4.2 Integration of Security in the PTPv2 Stack . . . 44
  - 4.2.1 Structure of the Security Module . . . 46
  - 4.2.2 Implementation Details . . . 51
  - 4.2.3 Verification and Validation . . . 56
- 4.3 IPsec Protection . . . 58
  - 4.3.1 Structure . . . 58
  - 4.3.2 Interfaces . . . 59

5 Measurement Setup **61**
5.1 Setup of the Hardware . . . 61
5.2 Packet Delay Variation Measurement . . . 62
5.3 Measurement of Clock Synchronization Precision . . . 63
5.3.1 IEEE 1588v2 Setup . . . 64
5.3.2 IPsec Setup . . . 65

6 Measurements and Results **69**
6.1 Influence on Measurement Results . . . 71
6.2 Packet Delay Variation Measurement . . . 72
6.2.1 IPv4 without Security . . . 74
6.2.2 IPv4 Authentication Header in Transport Mode . . . 74
6.2.3 IPv4 Authentication Header in Tunnel Mode . . . 76
6.2.4 IPv4 with Encapsulation Security Payload in Transport Mode . . . 76
6.2.5 IPv4 with Encapsulated Security Payload in Tunnel Mode . . . 79
6.2.6 IPv4 Encapsulating Security Payload and Authentication Header in Transport Mode . . . 79
6.2.7 IPv4 Encapsulating Security Payload and Authentication Header in Tunnel Mode . . . 83
6.2.8 Summary of Packet Delay Variation Measurements . . . 83
6.3 Clock Synchronization using PTPv2 . . . 84
6.3.1 PTPv2 over IPv4 without Security Extension . . . 85
6.3.2 PTPv2 with Native Security Extension Annex K . . . 86
6.3.3 PTPv2 with Authentication Header in Transport Mode . . . 86
6.3.4 PTPv2 with Authentication Header in Tunnel Mode . . . 88
6.3.5 PTPv2 with Encapsulating Security Payload in Transport Mode . . . 89
6.3.6 PTPv2 with Encapsulating Security Payload in Tunnel Mode . . . 90
6.3.7 PTPv2 with Encapsulating Security Payload and Authentication Header in Transport Mode . . . 91
6.3.8 PTPv2 with Encapsulating Security Payload and Authentication Header in Tunnel Mode . . . 92
6.3.9 Summary of the Synchronization Measurements . . . 93

7 Conclusion and Perspectives **95**
7.1 Conclusion . . . 95
7.2 Perspectives . . . 96

Literature **99**

Annex "K" – Cryptographic Acknowledgment **103**

Abbreviations

AES	Advanced Encryption Standard
AD	Authentication Data
AH	Authentication Header
BMC	Best Master Clock
CSV	Comma Separated Values
DDoS	Distributed Denial of Service
DoS	Denial of Service
DTSS	Digital Time Synchronization Service
DUT	Device Under Test
ESP	Encapsulated Security Payload
FCS	Frame Check Sequence
GNU	"GNU's Not Unix!", recursive acronym
GPS	Global Positioning System
HMAC	Hash Message Authentication Codes
ICV	Integrity Check Value
IKE	Internet Key Exchange
IP	Internet Protocol
IPsec	Internet Protocol Security
IRIG	Inter Range Instrumentation Group
KINK	Kerberized Internet Negotiations of Keys
MAC	Message Authentication Code
NAT	Network Address Translation
NIST	National Institute of Standards and Technologies
NTP	Network Time Protocol
OSI	Open Systems Interconnection
PDV	Packet Delay Variation
PKI	Public Key Infrastructure
ppm	parts per million
PPS	Pulse Per Second
PTP	Precision Time Protocol
SA	Security Association
SAD	Security Association Database
SDL	Specification and Description Language
SHA	Secure Hash Algorithm
SP	Security Policy
SPD	Security Policy Database
SPI	Security Parameter Index
TLV	Type Length Value
TTL	Time To Live
TTP	Time-Triggered Protocol
VPN	Virtual Private Network

1 Introduction

Networks used in industrial environments often have a need for time-referenced data. To provide a reliable time information service the nodes in the network need a common notion of time. In former times a lot of implementations developed proprietary protocols and dedicated wiring to achieve this goal, e.g., Inter Range Instrumentation Group (IRIG) standard time code B [Tel04] or Time-Triggered Protocol (TTP).

Newer approaches [IEE08, MMBK10] make use of already existing network infrastructures and use communication protocols which are already available. A combination, which is very common today, is the IP based communication together with Ethernet as the physical layer. Concentrating on one network technology reduced the costs for the required components. Field bus systems were designed to work with time critical applications. Control functions rely on this concept and make heavy use of them. By contrast, network infrastructures, which use Ethernet were not designed having these principles in mind. Packet based communication, such as Ethernet has a non-deterministic behavior. Without further improvements it cannot be used for real-time applications.

Modern clock synchronization protocols can help to overcome this obstacle and introduce high-precision clock synchronization for packet-based communication networks. The clock synchronization serves as a foundation for applications, which need accurate timing information, e.g., the arbitration of a field bus. High-precision clock synchronization is one of the technologies, which is the basis for real-time services for such applications. The clock synchronization protocols use regular messages to distribute timing information between the nodes in the network. The clock synchronization protocols make use of filter algorithms and control loops to compensate for delays introduced during the network traversal.

One of these protocols is the Precision Time Protocol [MMBK10], which offers high-precision clock synchronization in packet-based networks. The size of such networks can range from a few nodes to several hundreds of nodes. Until now the security of such networks was a topic, which was not very popular. Threats for networks were rare and the costs and complexity to implement security schemes for specific clock synchronization protocols were not rated high enough to justify an implementation. However, more and more operators of such networks have become aware of security and want solutions, which provide the necessary functionality even in these large networks. Recently the clock synchronization protocols introduced security features within the protocol specifications, e.g., version 2 of PTP (released 2008). Additionaly, in some cases networks already have schemes in place to secure the traffic. One of these solutions is the Internet Protocol security (IPsec) and it is one of the most widespread protocols to establish secure communication.

The goal of this work is to evaluate state-of-the-art security schemes for high-precision clock synchronization. On the security solutions provided by PTP a security analysis is performed. For users it is necessary to know the impact of the security scheme on the clock synchronization performance. The result of the analysis can be used by network operators to determine, if the additional security has influence on the quality of the clock synchronization.

1.1 Motivation

Security for high-precision clock synchronization in packet-oriented networks is a cutting edge topic, which is not covered by many reviews yet. In addition, the topics of clock synchronization and security seem to be natural enemies, since security adds delay and jitter. The synchronization highly cares about the time [TH10].

The demand for a combination of the two fields of knowledge, clock synchronization and security, is driven by the needs of modern computer networks. Concepts for both applications are available but the fusion of the topics is a task, which can hardly be anticipated. The demands of the network operators does not end at the fusion of the two domains, they want to reuse already existing solutions. An operator who is already using a certain security implementation might not be willing to introduce yet another security measure, it is more favorable for him to adopt the clock synchronization to fit the available network structure.

Currently little information is available about the impact of security on clock synchronization networks. The algorithms, which are used for protection and encryption, have a non-deterministic behavior. This leads to an additional influence, which is not accounted for yet, and can reduce the precision and therefore the quality of the clock synchronization.

Security within a network can be implemented on different layers. It might be an option of the application and therefore provided by an application layer. Another approach would be to integrate the security directly in the IP layer and make use of already existing structures. For proper design of systems clear indicators are required to select the best solution.

The impact of the influencing factors on clock synchronization is manifold and unforeseeable. Therefore, an implementation has to be verified via experiments, which in the end, add a noticeable change in the behavior of the clock synchronization stack. This is the goal of this thesis.

1.2 Goal of this Work

Within this diploma thesis a secure high-precision clock synchronization stack has to be implemented, which offers the possibility to securely synchronize nodes in a network. The protocol used for this application is the Precision Time Protocol, which is standardized in IEEE 1588v2. The effect of the implemented security has to be analyzed; which impact does it have on offset and accuracy of the clock synchronization?

Namely the following tasks and requirements have to be fulfilled:

- Security analysis of Precision Time Protocol Annex K
- Implementation of secure stack for clock synchronization

- Analysis of the implemented security
- Comparing IPsec with the developed security solution provided by IEEE 1588v2

This diploma thesis is structured as follows:

- Chapter 2 covers the state-of-the-art clock synchronization protocols, which are most commonly used. It gives a general overview about security in networks. Furthermore, it shows the technologies, which can be used to secure traffic in clock synchronization networks.
- Chapter 3 analyses the native security solution of the Precision Time Protocol and shows possible attacks and weaknesses. The different attacks are explained in detail together with the consequences which arise.
- Chapter 4 covers the design and implementation of a secure clock synchronization stack. It also introduces means to verify and validate implementations for standardization and testing of interoperability.
- Chapter 5 presents the measurement setup used in chapter 6 for all measurements. It holds information about the configuration of the components, which are needed to run the measurements.
- Chapter 6 shows the comprehensive measurement results, which were conducted to evaluate the implementations.
- Chapter 7 gives a resume of the work and gives an outlook on possible fields for further scientific research.

2 Technology Overview

This chapter gives a brief introduction on the topic of clock synchronization protocols for today's networks. The first part explains the need for clock synchronization followed by the two main protocols used in this domain, namely the Precision Time Protocol (PTP) and the Network Time Protocol (NTP). The second part focuses on the available options to secure the clock synchronization traffic within networks.

2.1 Application of Clock Synchronization

In today's world it is common to be able to set the clock of a computer to a specific time. One hundred years ago this task involved adjusting the local mechanic clock to a mechanic clock nearby. Even today many people have wristwatches, which are set manually. The accuracy that is achieved with this adjustment is usually within one or two minutes. Most people never set their clocks again until they miss a bus (typically half a year after they set their watch). After such an event they check their watch again only to realize that the time is already off by five minutes. A drift of five minutes over six months equals 27 parts per million (ppm). This value is quite good for temperature-stabilized watch. Typically clocks used in modern computers have an error of 100 ppm or more, which is a drift of at least four minutes and 32 seconds per month.

For day-to-day use such a drift does not mean a lot, but a consequence could be that an email seamingly arrives at the receiver before it got sent. The situation changes completely when the topic is changed to more time sensitive applications. These applications depend on reliable timing information, that needs to be sure that the time is in a certain interval to provide confidence. For example the Global Positioning System (GPS), it heavily depends on synchronized clocks. The position is calculated with the help of the signal delays between the different satellites. A difference of only 3 ns already results in a difference of approximately one meter between the displayed position and the actual position. It is the domain of clock synchronization, which makes such tasks even possible.

In the domain of computer science many models used assume a common notion of time. When something happens and this information is distributed to another entity, the event happened before the message arrives. For the applications, the basic assumption is always an increasing time. Therefore, when a message with the current timestamp is sent from one entity to the next entity the arrival is always later than the time included in the message. Making a leap from theoretical models to real networks reveals an interesting fact. Real clocks used in network nodes

run at different rates. A communication protocol, which adjusts the rate of two clocks to have a common notion of time, can operate within a certain range. This problem can be expanded to more than two network nodes, for example to a network consisting of hundreds of nodes. The goal of such a communication protocol is to align the clocks of multiple network devices as good as possible.

Modern computer use quartz oscillators and counters to produce intervals, which interrupt a running processor every few milliseconds. Such an interval is called a tick. Each tick leads to an increase of a variable, which is representing the actual time of the computer clock. This value is available for the use in other applications. If necessary the time can be adjusted to a specific absolute value, which is equal to setting the current time on a mechanic clock. The period of the tick can be adjusted in small steps to adjust the rate of the clock smoothly.

In Figure 2.1 the time of a local clock compared to a reference clock is depicted. To adjust the local clock to the time which is provided by the reference clock, the rate of the clock is adapted. The aim of the node is to adjust its local clock as good as possible to the reference time. In the diagram the local clock is slower than the reference clock – the ticks are "too big". The control algorithm adapts to the new situation and adjusts the rate to make the clock go faster. This procedure is done constantly to keep the local clock synchronized to the reference clock.

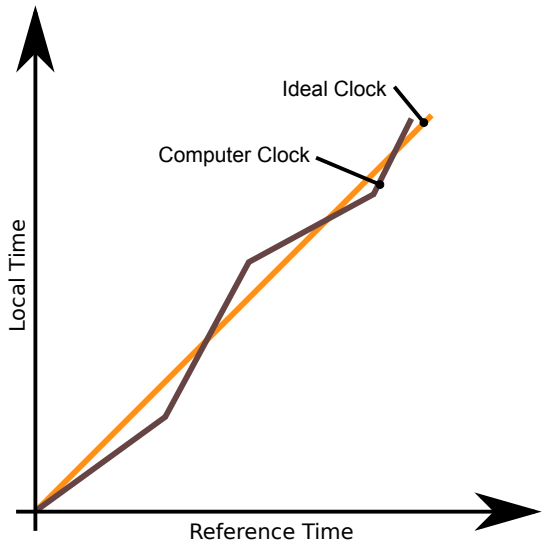

Figure 2.1: Comparison of ideal clock with computer clock

Security seems a little bit off topic for the domain of clock synchronization, but accomplishing this task can be vital for some applications. Using an already available open network to synchronize network components is very appealing but also threatening. Apparently a lot of things can go wrong when an attacker adjusts the clock values of nodes, which play well with this bad intentions. Areas of applications, which facilitate clock synchronization and are targets for attackers, can be telecommunication, substation automation, factory automation, etc. This diploma thesis concentrates on the security aspects and shows possible implementations, which can be used to secure a clock synchronization network using IP-based networks.

2.2 Protocol Overview

Protocols used to synchronize two or more clocks are trying to eliminate the offset between the local time and a distant clock. There are a variety of possible solutions for this problem. All of today's clock synchronization protocols follow a general approach where a client contacts a time server, which then responds with the current time. To increase the accuracy of the system the client has to determine the delay of the connection to find out the real offset of clocks. Especially, in big networks (such as the Internet) several factors can have a great influence on the delay, e.g., the time of day, the path which is used for a packet is not fixed, etc.

The protocols used for such a task have to handle a wide range of applications. In practice a small network for automation starts with only two nodes, which are to be synchronized. The size can easily extend to several hundred nodes in big production halls. Automation networks are typically the domain of IEEE 1588v2, a protocol discussed later in a subsection 2.2.2. Another common protocol used for clock synchronization is the Network Time Protocol (NTP), which is the topic of the next subsection 2.2.1. The public NTP network consists of several hundreds of servers and millions of clients connected to it. The National Institute of Standards and Technology (NIST) estimates 25 million clients are communicating directly with their servers [Mil06]. This number only includes the direct communication. There are many more subnetworks which are private and indirectly receive the time via intermediate servers. This number might be even bigger.

Reliability for clock synchronization networks is often solved with the help of multipath connections. Using multiple servers and different paths to these servers provides the necessary diversion for such an application. Such a scheme protects against hardware errors, broken network links, and faulty software. Also common for clock synchronization protocols is a hierarchical arrangement of the communication partner, e.g., a server/client or a master/slave scheme.

2.2.1 Network Time Protocol – NTP

The Network Time Protocol (NTP) (specified in [MMBK10]) is a protocol used for synchronizing clocks of computers in networks. It is designed for packet-switched networks and is one of the oldest clock synchronization protocols. Development of this protocol started in the 1980s with the latest version being released in June 2010 (NTPv4). The original design is from David L. Mills who is still maintaining the releases. This chapter provides an overview of the used topology, the structure of the messages, the clock selection algorithm and the clock filter algorithm, which are used in NTP.

To increase the reliability of the clock synchronization NTP uses many different servers and even completely different networks. When a lot of connections are established, at first it cannot be determined which offers the best conditions for the clock synchronization. This is where the clock synchronization algorithm performs its work. The algorithm tries to find the best configuration to operate in. Having found the optimal set of clocks for the synchronization, the clock filter algorithm is facilitated. It selects from the received data sets the best samples and applies mathematical algorithms to suppress errors observed due to network problems, outliers etc.

Network Topology of the Network Time Protocol

The structure of NTP is hierarchical with different levels of clock sources. The levels are called stratum and start with stratum-0, which is the best source available for the clock synchronization. With the final stratum level being 15. In NTP a stratum can be defined as the distance between a local clock and a reference clock. A computer getting its timing information from level n, is classified in level n+1. A generic classification can be made for the according to [Mil06]: Each stratum level consists of many hosts, which all offer the same functionality for clock synchronization and work as servers as well as clients. The duplication of the computers offers two benefits. Firstly, a higher redundancy, since if one of the computers is not accessible several others can fill the gap. Secondly, the distance between the computers can have great influence on the quality of the synchronization. Therefore, some computers are a better choice, e.g., they are nearby or have a very stable connection. A generic classification of the stratum levels can be made as follows:

- Stratum-0: This level is the source of the clock hierarchy. Stratum-0 are high-precision clocks, which are called reference clocks, e.g., atomic clocks, GPS clocks, etc. These clocks are usually not connected directly to the network. It is to use them as a source for a locally connected computer.
- Stratum-1: Clocks with a stratum-0 level are directly connected – not over a network connection – with stratum-0, e.g., a Pulse Per Second (PPS) output. Stratum-1 servers are commonly called time servers.
- Stratum-2: Computers, which are part of this level, get their timing information form stratum-1 servers. This level uses several stratum-1 servers as source and chooses only the best amongst them. Obviously wrong servers are dropped automatically. Stratum-2 devices communicate with other stratum-2 devices to build a stable group for timing information.
- Stratum-3 to 15: This level employs the same functionality as stratum-2, but instead of stratum-1 as source it depends on several stratum-2 sources. The maximum depth of the network, which is supported, is level 15 (stratum-15).

An illustration of the levels is given in Figure 2.2.

Protocol Structure of the Network Time Protocol

In NTP the time is passed on from one source to another as described in subsection 2.2.1. When a client is synchronizing to a server in the network, several packets are exchanged in a request-response procedure.

The client starts the procedure when it sends a request to the server. The request contains the local time called origin timestamp. The server which is receiving this packet adds a local time when the message is received in the field receive timestamp. The packet is then prepared to be returned to the sender. When the packet is sent, the local time is inserted in the field transmit timestamp. The receiver of the reply stores the time of arrival. With the timestamps the receiver can estimate the time, which the packet needed to travel from the local source to the server and back. The total delay of a packet is the travel time of the packet minus the residence time at the server. It is assumed that the delay between the two hosts is symmetrical and therefore the delay from one host to the other is the half of the total delay.

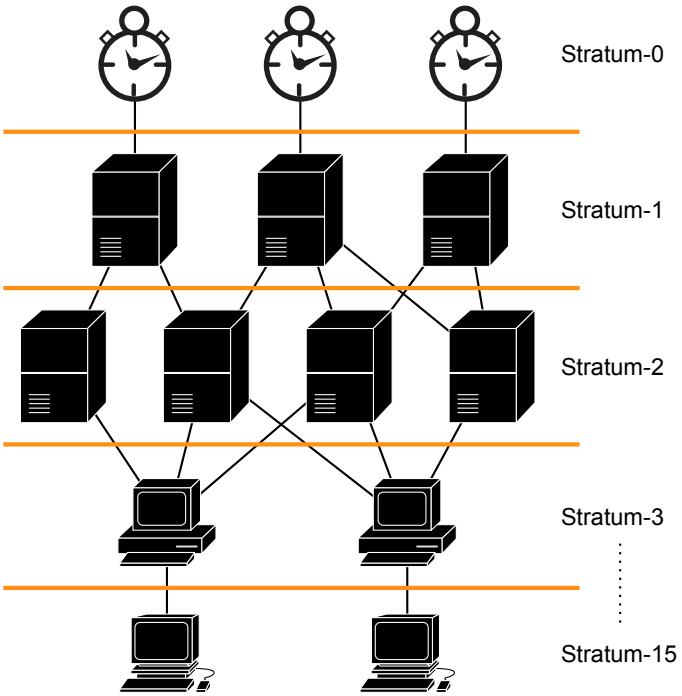

Figure 2.2: Structure of NTP networks with different stratum levels

The timing information in the packet is also used to find the time offset between both hosts. This offset is corrected with the delay, which is determined before. The timing information from servers are not used immediately, several messages are exchanged and sanity checks are performed to assure a certain quality of the data.

When an NTP stack is started, it reads from the configuration file to find the sources for the synchronization. If the stack was already running, it provides the gathered statistical data, which are considered for the current execution. After this initializing phase the stack is up and running and starts exchanging packets with the servers provided at startup. Usually, the stack needs five minutes to apply all the algorithms and optimizations and to accept the timing information from a server. After this procedure the NTP stack is constantly updating the local clock to supply the applications of the system with this information.

The communication between the hosts is based on NTP messages, which are transported over UDP and use port 123. The structure of the NTP packet can be seen in Table 2.1. The following fields are included:

- Leap Indicator: This 2 bit value indicates an impending leap second. Four states are possible – no leap second, not synchronized, add one second, and subtract one second.
- Version: This value indicates the used NTP version, the most recent version is NTPv4.
- Mode: This field indicates the mode of the sender. The most common values are server and client mode.
- Stratum: This variable defines the stratum level for the clock. Value 1 is reserved for reference clocks and 2-15 is for the different possible levels.

- Poll: This field is of type 8 bit unsigned integer, which indicates the maximum interval between two successfully exchanged messages. The value is interpreted as second and set to the nearest power of two, the poll interval itself is 2^{Poll}, and starts with 4 seconds and goes up to approximately 36 hours.
- Precision: The precision of the clock is of type 8 bit signed integer indicating the precision of the clock. It is measured in seconds and is equal to the nearest power of two, i.e., a value of -18 for the precision is analog to a precision of about 4 µs (2^{-18}s). This value is measured when the system is started up and represents the minimum over several attempts to read the system clock.
- Root Delay: This value indicates the total round-trip delay in seconds to the reference clock.
- Root Dispersion: This value indicates maximum error relative to the reference clock, in seconds.
- Reference ID: This value is a clock identifier and is depending on the stratum level of the clock. Stratum-0 clocks have a four character ASCII code assigned that is called a kiss code. Also stratum-1 devices are assigned 4 byte left-justified, zero-padded ASCII string [Mil06]. For higher stratum levels the ID consists either of the IPv4 address or, if IPv6 is used, the highest 4 byte of the address or a MD5 hash of the IPv6 address.
- Reference timestamp: The reference time denotes the local time of the node.
- Origin timestamp: This value holds the time at which the request to the server was sent.
- Receive timestamp: This timestamp contains the arrival time on the server side.
- Transmit timestamp: The transmit timestamp is the time when the message departs from the server to the client.
- Extensions field: This optional field is used by the Autokey public key cryptographic algorithms [HM10], more information can be found in section 2.3.1.
- Message Authentication Code (MAC): This is an optional value, which consists of the key ID and a message digest. It is only present when an authentication for the NTP message is used.

Clock Selection Algorithm

To ensure a reliable synchronization, NTP makes use of multiple time sources. These are redundant servers from the same stratum level and normally selected to have multiple different network paths available. At the beginning, when the connections are established, the node cannot distinguish between nodes which distribute an imprecise timing information and such nodes, which provide correct timing information. It is important that the algorithm used for this scheme is robust and discards the false timing information.

David L. Mills provides in his book a proof for the following statement:

> "... the true offset θ of a correctly operating clock relative to UTC must be contained in a computable range, called the confidence interval ..." [Mil06, page 47]

Type	Length
Leap Indicator	2
Version	3
Mode	3
Stratum	8
Poll	8
Precision	8
Root Delay	32
Root Dispersion	32
Reference ID	32
Reference Timestamp	64
Origin Timestamp	64
Receive Timestamp	64
Transmit Timestamp	64
Extension Field 1	variable
Extension Field 2	variable
Key ID	32
Message Digest	128

Table 2.1: Structure of a NTP packet, with type and length (in bit) of the fields

Marzullo and Owicki [MO85] developed an algorithm, which is able to find an intersection interval that contains the correct time, for m clocks with a given confidence interval and no more than f clocks providing incorrect input. The result of the algorithm is an interval, which provides the smallest possible set of intervals that contains at least m-f of the provided confidence intervals.

Figure 2.3 depicts the algorithm when used with four clocks. The example uses for clocks A, B, C, and D with their respective confidence interval and an offset in the middle of the correctness interval. The true offset of a host can be found anywhere in the correctness interval. Based on the proof that the true offset must be contained in the confidence interval and all provided timing values are correct, a non empty interval exists, which contains points in all four confidence intervals. As it can be seen in the diagram, this assumption does not hold true for this example. Therefore not all of the clocks provide correct values. If it is assumed that one of the clocks is the source of the false information, e.g., D, it is possible to form a non empty interval. When this approach does not work out, the next step would be to discard two clocks and so on.

The Digital Time Synchronization Service (DTSS) is based on this principle. It searches for the smallest possible intersection interval that contains at least one point in m-f provided intervals, m is again the total number of intervals and f is the number of hosts, which provide imprecise values. This procedure works as long as $f < \frac{m}{2}$. For the given example, this algorithm finds the interval marked with the label "Interval DTSS", with D being identified as wrongdoer.

The interval, which is found by the algorithm of the Digital Time Synchronization Service, does find the smallest interval containing the correct timing information, but it is not clear which value in the interval has to be chosen. A simple approach would be to take the center of the found interval that would disregard a lot of statistical information, which results in a large jitter. This is confirmed by experiments. Therefore, NTP uses a modified version of the DTSS algorithm. The modified algorithm uses at least m-f clocks where all the midpoints have to lie within intersection

interval. Thus, this algorithm finds the smallest interval m-f, which contains at least m-f centers. The result of this modification finds the interval labeled as "Interval NTP". The interval includes the calculated time for clock C.

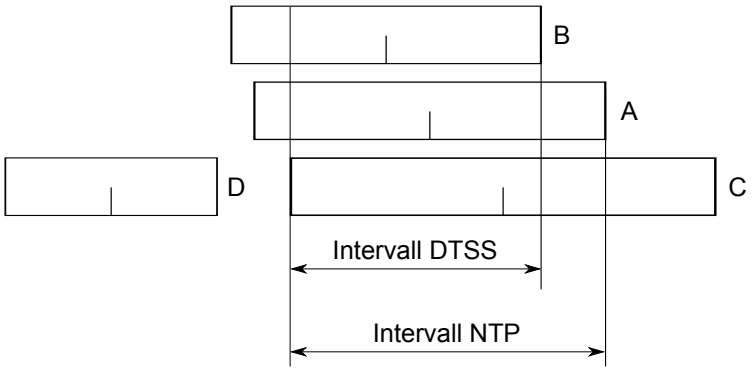

Figure 2.3: Selection of NTP intersection interval. Clocks with an offset and a confidence interval.

2.2.2 Precision Time Protocol – IEEE 1588v2

The Precision Time Protocol (PTP) is used for high-precision clock synchronization and is standardized in IEEE 1588v2 [IEE08], currently available in version two release 2008. IEEE 1588v2 is designed to supply higher accuracy than what is achievable with NTP.

This protocol has a hierarchical structure with a master slave architecture. The master distributes the timing information to the slaves. Messages in PTP can be divided into two groups, timed messages with accurate timestamps, called event messages, and general messages, which do not require timestamps. The event messages are used to measure delays and provide accurate timestamps for the calculation performed by the nodes.

The PTP protocol uses different types of messages for synchronization and management of the PTP nodes. These messages are divided into two groups. First, the type of messages which are time-critical and have direct influence on the clock synchronization accuracys these are called event messages. Second type are general messages, which do not interfere with the accuracy of the synchronization.

The synchronization between a master and a slave uses several types of messages, namely these are Sync, Follow_Up, Delay_Req, and Delay_Resp messages. A master periodically sends Sync messages, the slaves take the information from this message and use it to set the local clock. Clocks can also make use of the Follow_Up message. This type of message is not time critical and is used to correct received timestamps afterwards. This approach uses two messages instead of only one for the synchronization and therefore needs double the bandwidth in the network.

For further enhancement the delay request-response mechanism can be used, depicted in Figure 2.3. It measures the mean path delay between two PTP ports. After all four messages Sync, Follow_Up, Delay_Req, and Delay_Resp are exchanged, the slave knows four timestamps marked with t_1, t_2, t_3, and t_4. t_1 and t_4 are measured using the time of the master and t_2 and t_3 are measured with the time of the slave. The mean path delay can be calculated with $meanPathDealy = \frac{(t_2-t_1)+(t_4-t_3)}{2} = \frac{(t_2-t_3)+(t_4-t_1)}{2}$. This value is the mean value of $t_{ms}+t_{sm}$. If the delay asymmetry between the master-slave and slave-master is known, this value can be added to the control algorithm to get the best result for the synchronization.

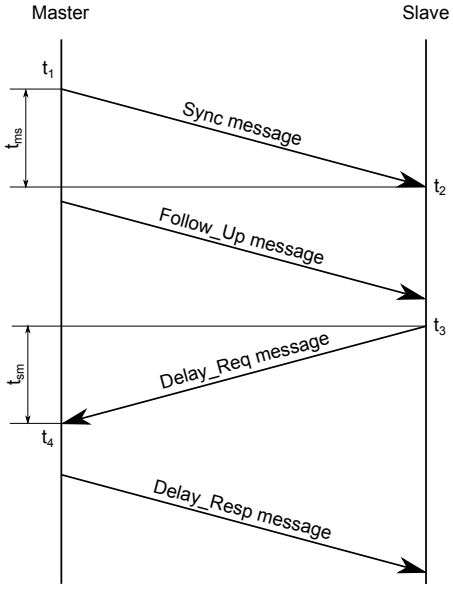

Figure 2.4: Synchronization between a master and a slave node

Network Topology of the Precision Time Protocol

PTP is a protocol which works on the application layer and utilizes UDP for transportation. Two ports are occupied – 319 for event messages and 320 for general messages. PTP uses a hierarchical structure, with nodes in master or slave mode. Slave nodes get timing information from master nodes. It is also possible to form subnets within an existing clock synchronization network. Complex network arrangements can be implemented, an example for such a network is depicted in Figure 2.5.

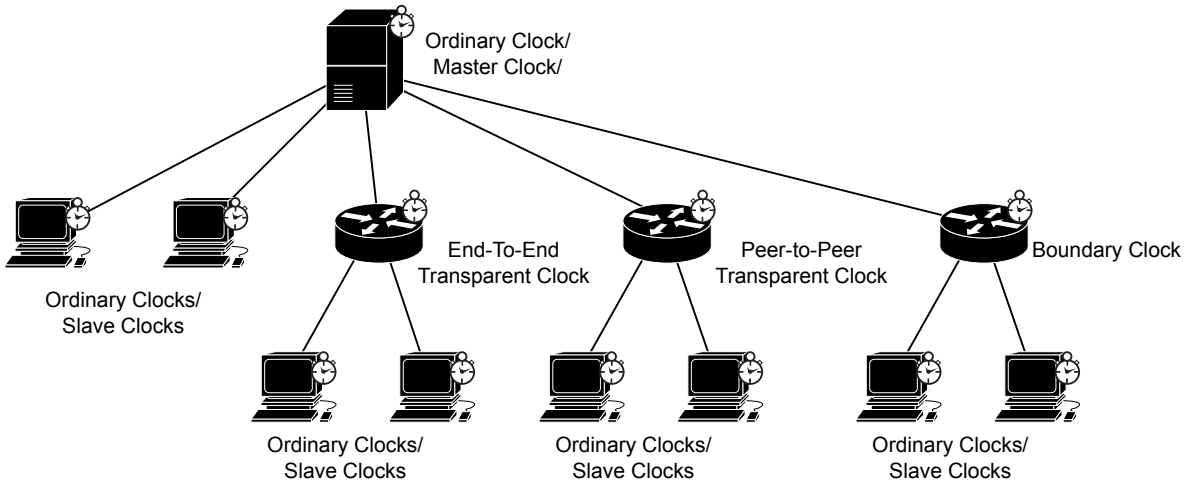

Figure 2.5: PTP network topology

Aside from the hierarchical master and slave network structure – where nodes are called ordinary clocks – there are also other different types of PTP devices i.e. boundary clocks, end-to-end transparent clocks, peer-to-peer clocks, and management nodes, which are mainly used for building up the hierarchy and connect the hosts. The properties of the different devices are as follows:

Ordinary Clocks

These devices are network components with a single physical port, which is used for the two logical interfaces, the event messages and the general messages. This device is part of a clock domain and has one running instance of the PTP stack. Such a device can be either a master device or a slave device. When the device is in a master state and the clock is a grandmaster (the highest master in a group) it is typically synchronized to an external source, e.g., the GPS system.

Boundary Clocks

These components are typically used as network elements and are not connected to any application. This type of device only cares about PTP devices for traffic not belonging to PTP it behaves like a normal network device, e.g., a bridge, repeater, or router. Boundary clocks have several physical ports in contrast to ordinary clocks. Similar to the previous clock type, each interface has two logical interfaces, an event port and a general port. All ports use the same local clock for synchronization and each running instance of the PTP stack can resolve the state of all ports to decide which port is used for the clock synchronization. PTP messages used for synchronization or establishing the hierarchy are not forwarded to the other ports. Only management messages are forwarded to the designated node.

End-to-End Transparent Clocks

These network elements perform as ordinary network devices, with the ability to change fields in PTP message. This type of device can compensate for the delay that is introduced by traversing the end-to-end transparent clock. During the traversal of the message in the network the retention time within a network device is not deterministic, e.g., due to queues. To overcome this flaw the end-to-end transparent clock determines the time the packet needed to be received on one port and sent out on another port. This time difference is inserted into the correction field of the PTP header, see Table 2.2. Most relevant for security is the fact that in order to produce valid packets the device also has to update the cryptographic checksums.

Peer-to-Peer Transparent Clocks

These devices are similar to the end-to-end transparent clocks. The subtle difference is that the peer-to-peer clock directly connects the PTP nodes. With this setup it is possible to measure the link delays for each port, because the delay depends on the ports involved in the connection. The delay can vary between a connection of port one to port two, or port one to port three. Therefore this device compensates for the complete path delay of a PTP message.

Management Nodes

Devices belonging to this group are nodes that have an interface to manage PTP nodes. This can be an interface for humans or for applications. Thus, it has at least one interface for the communication with the PTP network. This management interface can be coupled with every clock type mentioned so far.

Protocol Structure of the Precision Time Protocol

The communication between the network devices uses PTP packets. These packets are composed of a common header and message specific part. All PTP messages can be extended by one or more Type Length Values (TLVs). All these TLVs have unique identifier. When a PTP message contains TLVs which cannot be processed by the PTP stack, such TLVs are ignored. The structure of a packet with an preceding transport header from the transport layer is depicted in Figure 2.6.

Figure 2.6: PTP message structure

The structure of the PTP header can be further divided as depicted in Table 2.2. Each message contains the following fields:

- transportSpecific: This field may be used by lower layer transport protocols, the specification is defined in Annex F of IEEE 1588v2.
- messageType: The field specifies the type of the message, e.g., Sync, Delay_Req, Follow_Up, Signaling, etc.
- versionPTP: As of this writing two different versions of PTP exist, which can be distinguished by the version number, namely these are the initial version labeled with 1 and the version released in 2008 labeled with 2.
- messageLength: This value reflects the length of the message in byte. It starts with the first byte of the header and includes all optionally appended TLV to end at the last byte of the last TLV.
- domainNumber: It is used to group clocks. Trough that option the network can be divided into several logical subnets within the existing clock network.
- flags: The different flags reflect the options, which can be used in the network, e.g., use security, send only unicast messages, use PTP profile one or two, etc.
- correctionField: This field holds the accumulated delays, which occurred during the network traversal. The timing value is inserted in nanoseconds.
- sourcePortIdentity: It represents a unique identifier for all nodes participating in the network.
- sequenceId: The sequence ID is used to find corresponding packets. Several types of PTP messages are sent in a request/reply procedure, to match these packets the sequence ID is used.
- control: This field is provided for compatibility to PTPv1; for PTPv2 it is deprecated.
- logMeanMessageInterval: This value specifies the message interval for the periodic communication between master and slave.
- suffix: After the header a PTP message and an arbitrary number of Type Length Values (TLV) can be appended.

Type	Length
transportSpecific	4
messageType	4
reserved	4
versionPTP	4
messageLength	16
domainNumber	8
reserved	8
flags	16
correctionField	64
reserved	32
sourcePortIdentity	80
sequenceId	16
control	8
logMeanMessageInterval	8
suffix	variable

Table 2.2: Structure of a PTP header with type and length (in bit) of the fields

A PTP message extends the common header with the fields specified in IEEE 1588v2. An example for such a message would be the synchronization message, depicted in Table 2.3.

Type	Length
PTP header	272
originTimestamp	80

Table 2.3: Structure of a PTP synchronization message with type and length (in bit)

Best Master Clock Algorithm

The Precision Time Protocol uses a master slave structure, therefore slaves are synchronizing to a master. A master is not a fixed position, every node can either be a master or a slave. The Best Master Clock (BMC) algorithm is a procedure that determines which node has the best clock and is therefore preferred to be selected as master. Every node in the group is running this distributed algorithm. This process is not a one-time event, but it runs continuously and adapts to changes in the system, e.g., properties of a node change, a new node is added, or an existing node is removed.

The algorithm itself can be divided into two parts:

1. A comparison of clock properties that determines which of the clock ports is better.
2. A process suggesting a state for the ports which are available.

The decisions are based on the information, which are collected with the receiving of announce messages.

Comparison Algorithm

The compare algorithm of the PTP stack compares datasets, the representation of available clocks in the network. The decision is based on a priority list with the following order:

1. Priority 1
2. Class
3. Accuracy
4. Variance
5. Priority 2
6. Identity

The lowest priority has the identity of a clock. This identity is based on the MAC address of the node. When two clocks share the same dataset the decision of which is the better clock is based on the MAC address. The clock with the lower MAC address wins the election.

State Decision

The state decision of each clock port is determined by an algorithm during the BMC. The dataset, of a clock is compared with the current clock port. The comparison algorithm results in a state for this specific port, then the next port is chosen until all ports of the local device are served.

2.3 Security

Securing information has a long history. Also in todays modern networks a proper protection is an important issue and is needed more than ever. Over the years many terms and definitions were established.

One of the most famous examples might be Caesar's cipher. As the name suggests it was used by Caesar to protect his orders or messages with tactical information for his generals. Some simple terms, which are used in the domain of security, can be explained with the help of this traditional story.

In ancient Rome, Caesar had the need to get information to his generals, yet he did not want his enemies to see it. He would start with writing the message, which would be transported by the messenger. This text is called a cleartext. If a villain or spy from an adversary can get hold of the message, it would be easy to read the information. To overcome that obstacle Caesar used a rule to transform the cleartext into a new sequence of characters. This transformation rule is called a cipher and the result is the ciphertext. Caesar used a substitution algorithm where a letter from the cleartext is replaced with a letter several positions down the alphabet. The process of transforming cleartext into ciphertext is called encryption and today many different algorithms are available to perform this task. The recipient got the encrypted note from the messenger and then he performed the converse procedure to get a cleartext again. This procedure is called the decryption.

This concept is very simple and does not offer a lot of confidence. The information, which is transported by the messenger, could have been modified. Also it is not guaranteed that the message was written by Caesar. The same assumption holds true for answers as well. Caesar cannot be sure that the answer was written by one of his trusted generals or a foe. Therefore, concepts and definitions have been derived, which are used in state-of-the-art applications.

Modern computer security defines goals for protecting an asset. The security goals consist of the following parts: confidentiality, integrity, availability, authentication, access control, and non-repudiation. These terms and definitions are important for the understanding of the concepts and examples used in this section. The order is alphabetical and does not reflect any priority, this is just for the convenience of the reader who might need to lookup some explanations.

Access Control Essentially access control defines who is allowed to do what in a system. Access control is ubiquitous, for example some users are allowed to delete files and another users are allowed to use the network connection. A more day-to-day example would be the key to a house or car, only a person with the key is allowed to access. This service makes use of several other services defined in a secure environment, namely these are: authentication, confidentiality, integrity, and availability.

Authentication This service is affiliated with the challenge of identification. In networks, the purpose is to identify a communication partner. Entities, which establish a communication, have to authenticate each other. After this procedure each entity is sure with whom it is communicating.

Availability A system or a part of it, e.g., a device in the system, is available when it is in a functioning condition, called uptime. The opposite would be unavailability. This is the time when the system or device is in a state where it is not functioning, called downtime. The availability can be specified as ratio of uptime to the sum of uptime plus downtime. The result of this ratio is typically given in the form of a decimal value, e.g., 0.9998, which means 99.98% uptime and 0.02% downtime.

Confidentiality The service of confidentiality offers the possibility to keep information private from third parties. Only authorized entities are able to gain access to protected data. Confidentiality can be provided in different ways, it can range from a physical protection of a device up to cryptographic algorithms executed in software.

Encryption The process of encrypting is a transformation from plaintext into an unreadable stream of characters. The transformation takes the plaintext as input, applies an algorithm and the output is the encrypted ciphertext. Encryption is the base needed for the service of confidentiality.

Integrity Integrity is a service, which ensures consistency. These functions are specifically designed to prevent intentionally and unintentionally modification of the data and detect such an attempt. For example, within Ethernet networks the integrity of frames is ensured by a checksum called Frame Check Sequence (FCS). To ensure the integrity of the data, cryptographic hash-functions can be used. The output of such functions is a hash-value, which is often also called Integrity Check Value (ICV).

Non-repudiation This is a service which prevents the sender of an information from denying the action it has performed. Thus, it guarantees that an information is tied to a specific sender. To provide the feature of non-repudiation, two properties have to be fulfilled. Firstly, the integrity of the data has to be verified, and secondly, the origin of the data has to be ensured.

2.3.1 Network Time Protocol-Security Extensions

The NTP security model is designed to work in mixed environments. It supports multi-level security environments and makes use of symmetric key cryptography, public key cryptography, and unsecured connections. The security concept of NTP makes several assumptions for the environment:

- The public key algorithms introduce an additional jitter, thus the performance of the synchronization is degraded.
- Depending on the operation mode, a server might not be able to store state variables for all clients. This information can be recovered at the arrival of a packet from a specific client.
- A reliable system clock provides the ability to implement a restriction on the life-span of cryptographic data. Therefore the entities, which exchange data for synchronization, have to establish a trusted relationship.
- The security from the client point of view depends only on public values, which can be transmitted over public networks. Private values are not allowed to be passed on, except if a trusted agent ensures a secure transport from one host to the other host.

The base of the NTP security model is the requirement for proofing the authentication to applications, which depend on the timing information. Starting with version three of NTP, functionality is included to authenticate specific servers with the help of symmetric key cryptography. The authentication can only be assured when each NTP server in the communication path can be authenticated. The authentication is applied to a server and not to the network gear, located between servers, e.g., routers, switches, etc. The authentication is done on a stratum level base. A higher stratum (lower number on the stratum level, e.g., stratum-2) authenticates the next lower stratum level (higher number on the stratum level, e.g., stratum-3).

Secure Groups in the Network Time Protocol

The NTP secure group is a subnet within the NTP network with a consistent security model. Each computer is able to authenticate other members within this group using cryptographic algorithms. Based on the hierarchical structure of NTP, also a secure group follows this approach. Every host within this secure group can verify the identity of direct descended server. The host can traverse the server hierarchy upwards and also ensure that the next server is valid. The system uses secure communication between the hosts; it is assumed that every member of such a group is in the possession of a secret group key. A computer can be a member of multiple secure groups, each of them with a dedicated key. The protocol and the algorithm, which is used in this process, is called an identity scheme.

Autokey Security Protocol

For secure communication NTP facilitates the Autokey protocol [HM10]. The base for the Autokey protocol is the Public Key Infrastructure (PKI) cryptographic functions of the OpenSSL library. The Autokey protocol also supports timestamped digital signatures as well as X.509 certificates for the authentication. To detect replay of messages a sequence number is included.

A Public Key Infrastructure uses public key cryptography and consists of several parts. The first piece is a server, which provides the connectivity needed by all participating communication partners. A Certificate Authority (CA) is needed to bind public keys to identities of the communication partners. The last part is the maintenance of the system, which has the ability to create, revoke, and manage certificates for the identities. For the communication between the entities, the Certificate Authority acts as a trusted third party, which manages the policies of the connections.

Autokey uses MD5 [Riv92] Hash Message Authentication Codes (HMACs) to detect the modification of a message. MD5 is also known as a one-way hash function. This function takes on the input side data with a variable length and provides an output with a fixed length, called a hash-value. The result is a fingerprint of the data and it is computationally impracticable to find two different inputs, which generate the same output. The incident of finding such input strings is called a collision. The most common use for such functions is to provide digital signatures or integrity protection for data.

The advantage of the Autokey protocol is the ability to divert attacks and simultaneously preserve the integrity and the accuracy of the clock synchronization functions [Mil06]. The protocol has the ability to make a preliminary authentication of a computer even when no reliable timing information has been verified yet. After the identity of a server is validated, signatures are verified and the timing information is available, the Autokey protocol has achieved its goal.

In NTP version 4, extra fields can be added to a message. The extension field consists of a 16 bit type, a 16 bit length, and variable data field and is located between the NTP header and the MAC. The example in Table 2.4 shows a extension field with several parameters.

Type	Length
Field type	16
Field length	16
Association identifier	32
Timestamp	32
Filestamp	32
Data length	32
Data	variable
Signature length	32
Signature	variable
Padding	variable

Table 2.4: NTP extension field for security

When a computer synchronizes to a server, which supports security, it uses an extension field, which incorporates a digital signature and a timestamp. After the verification of the server the host starts the synchronization process. Messages, which are replayed or do not pass a sanity check are discarded.

2.3.2 Precision Time Protocol-Security Extension

The PTP security extension defined in Annex K of the IEEE 1588v2 standard provides group source authentication, message integrity, and replay protection for PTP messages. The security of IEEE 1588v2 consists of two mechanisms:

- An integrity protection, which is based on Message Authentication Codes (MAC). The MAC is used to verify that a message originated from a trusted source and was not modified during the transport in the network. Additionally, it prevents the replay of a message with replay counters.
- A challenge-response procedure, used to authenticate new sources and to keep already established connections up to date.

The security extension does not provide non-repudiation or confidentiality services. There is a simple reason for that; PTP follows the concept of time as a common good and therefore it does not have to be confidential.

The protection of a message depends on the Integrity Check Value (ICV). The ICV is a MAC applied to the message. The actual algorithm that is used, is determined by the SA. The result of the MAC calculation is appended to the message and it protects against the modification of a packet. Any modification of the message can be identified because the receiver fails to verify the appended ICV with the reference ICV calculated over the actual message.

The replay protection consists of two attributes, the increasing replay counter and a random lifetime ID. Both are also part of the SA. The replay counter is incremented by two every time used on a sending process. A receiver has to verify that the received lifetime ID matches the lifetime ID stored in the incoming SA and that the received replay counter is larger than the one saved in the SA.

The security extension supports three different types of messages which are called Type Length Values (TLV):

1. Authentication TLV
2. Authentication Challenge TLV
3. Security Association Update TLV

These types are used in different stages of the communication flow and are explained within the next subsections.

The nodes participating in the communication share symmetric keys. These keys can be shared by a small group of nodes or by a complete domain. The distribution of the keys can be done either manually or with the help of key management protocols. Annex K supports both possibilities but does not specify the key distribution process itself.

A connection between the communication partners within the network relies on Security Associations (SA). The SA consists of several information used to verify that the packet is received from a trusted source. Another advantage is that these connections are uniquely identifiable, e.g., the source port of the sender, the source protocol address of the sender, the destination port at the

receiver. Every node maintains SAs for incoming and for outgoing connections. SAs can be used for single receiver but also for multiple receivers. In case of multiple receivers the different nodes have copies of the SA, which only differ in the value of the replay protection counter. The copies of the receivers are all smaller than the replay counter of the originator. When a single SA is used for all outgoing connections, one advantage can be noted. The replay counter of the single SA is increased more often. Therefore, the update frequency of the SA is higher. This results in a lifetime ID which is changed more often.

An SA is communicated from the sender to the receiver (or the receivers, in case of group communication to multicast). The sender decides if it creates a single SA for all outgoing connections or it creates a SA for each connection being made with the different destinations. This decision can be made depending on the individual requirements of the implemented network. The implementation differs only on the sender side in the PTP network. If a single SA is used for all destinations, the replay counter is increased with each sending of a message and an update of the SA is necessary much earlier. For an implementation with different outgoing SAs, the outgoing table is bulkier than in the single SA solution. From a receivers point of view, there is no difference which technique is used.

Securing the Traffic Flow

The primary goal of Annex K is the protection of the traffic between the participating entities. This protection is based on the Authentication Type Length Value (TLV), which uses a Integrity Check Value (ICV). The ICV is the result of applying a Message Authentication Code (MAC) function to the PTP message.

The authentication TLV is depicted in Table 2.5, which shows the structure of the TLV. The *tlvType* identifies the type of the TLV, in this case AUTHENTICATION. It is followed by the *length* field, which holds the length of the TLV minus 4 byte (*tlvType* plus *length*). The next field is the *lifeTimeID,* which is used for replay protection. This random number is specified by the SA, which is used for the communication. The *replayCounter* field is used as a replay counter, and incremented by two on each use [IEE08, section K.2]. The *keyId* is used to select the shared secret and the *algorithmID* is used to select the algorithm, used for the calculation. At the end of the TLV, the result of the cryptographic integrity protection is inserted together with a possibly needed padding to achieve a fixed TLV length. For an implementation according to IEEE 1588v2, two calculation schemes have to be supported, HMAC-SHA-1 and HMAC-SHA-256 [NIS02]. Additional schemes can be implemented but are not specified by IEEE 1588v2. Both hashing algorithms produce a result which is truncated and inserted in the ICV field (last field of Table 2.5). HMAC-SHA-1 is truncated to 12 byte and HMAC-SHA-256 is truncated to 16 byte. The padding, which is inserted before the ICV is only present when HMAC-SHA-1 is used and has 4 byte to achieve a constant TLV length.

Generation of Authentication TLV

The following procedure is needed to generate an Authentication TLV for a message, which is protected by an outgoing Security Association (SA):

1. The secret key is defined by the key ID, stored in the outgoing SA and used as a key for the hash algorithm.

Type	Length
tlvType = AUTHENTICATION	16
length	16
lifeTimeID	16
replayCounter	32
keyID	16
algorithmID	8
reserved	8
pad + ICV	128

Table 2.5: Structure of an IEEE 1588v2 authentication TLV with type and length (in bit) of the fields

2. The hash algorithm is specified by the algorithm ID and, together with the key ID, it is used to calculate the Integrity Check Value (ICV). The key table is indexed by the key ID and contains also the corresponding algorithm ID for a key.

3. The hash algorithm determined by the algorithm ID and the secret key is used to compute the actual ICV. The ICV calculation includes the first byte of the PTP header and ends with the last byte of the authentication TLV, thus it also includes the ICV field. During the calculation of the hash, the fields of the ICV are set to zero, and are included in the message digest.

Verification of Authentication TLV

The subsequent procedure is followed to verify a received message appended with an Authentication TLV and protected by an outgoing Security Association (SA):

1. Check the key ID supplied by the authentication TLV; if the key is unknown the message fails the ICV check and it is discarded.

2. The algorithm ID supplied by the authentication TLV has to be checked against the key list and must match the algorithm ID stored together with the key.

3. The hash calculation is applied to the message with the corresponding key ID and algorithm ID from the authentication TLV. For the calculation, the original ICV inserted in the TLV is replaced by zero. The calculation starts with the first byte of the PTP header and ends with the last byte of the security authentication TLV. The result of the calculation is compared to the value of the ICV received. A message passes the check when the two hashes are equal, if they do not match the message is discarded.

For sending and receiving messages a security association look-up is performed. The incoming messages are checked against the incoming security association table and for the outgoing messages the check is performed against the outgoing security associations table. This message is then matched to an applicable entry in the incoming security association table, where the source port ID and the source protocol address are compared to the received values. If no viable Security Association (SA) is found the message is discarded, otherwise the search returns the trust state of the SA. There are three trust states defined:

1. Trusted

2. Untrusted
3. Challenging

Trusted relations are established connections, which have already performed a challenge-response procedure and no timeout has occurred. Untrusted connections are connections, which have timed out and marked for removal from the table. Connections in a challenging state have started a new connection and are waiting for a response from the communication party.

In addition to the state of the connection, the key ID can match two values, key ID and next key ID. In a nutshell, if an incoming connection does not match either of these fields the message is discarded. Additional information on this topic can be found in subsection2.3.2.

The next chapter gives an overview on howto establish a valid connection between communication partners.

Establishing New Security Associations

Before two entities can exchange information they have to perform a mutual authentication, to ensure their identity. IEEE 1588v2 is using a challenge-response procedure to perform the authentication. The challenge-response procedure includes the exchange of three different message types:

1. Challenge-request
2. Challenge-response-request
3. Challenge-response

The Authentication Challenge TLV is depicted in Table 2.6, which shows the structure of the TLV.

Type	Length
tlvType = AUTHENTICATION_CHALLENGE	16
length	16
challengeType	8
reserved	8
requestNonce	32
responseNonce	32

Table 2.6: Structure of an IEEE 1588v2 Authentication Challenge TLV with type and length (in bit) of the fields

The tlvType identifies the type of the TLV, in this case AUTHENTICATION_CHALLENGE. It is followed by the length field, which holds the length of the TLV minus 4 byte (tlvType plus length). The next field is the challengeType, which can be set to: challenge-request, challenge-response-request, or challenge-response. The requestNonce and responseNonce are values, needed to establish a connection. A chronological overview of the process is depicted in Figure 2.7 and the procedure itself is explained with the help of the following example.

In Figure 2.7 Node 1 sends a message secured with an authentication TLV, which is received by Node 2. Node 2 performs a look-up in the incoming Security Association (SA) table and does not find a corresponding SA. The message is immediately discarded and a challenge-request for Node 1 is triggered.

Node 2 inserts a random number into the requestNonce field and sets the responseNonce to zero. The TLV where the values are inserted is shown in Table 2.6. The ID and the nonce, which was sent, is stored for later use.

Node 1 receives the challenge-request and triggers a challenge-response-request. The received requestNonce is copied to the responseNonce and a new random number is inserted into the requestNonce field. This time, Node 1 stores the ID and the requestNonce of the sent message.

Next in the chronological overview is the receiving of the challenge-response-request at Node 2. The challenge-response-request contains a new requestNonce from Node 1 and a responseNonce, which was sent as a requestNonce in the challenge-request message. Node 1 matches the received responseNonce with the saved requestNonce. On a mismatch of the two nonces, the message is discarded immediately. If the two nonces match, Node 2 prepares a challenge-response message, it copies the received requestNonce and sets the responseNonce to zero. The last action from Node 2 within this procedure is to set the newly established connection to trusted.

Node 1 receives the last message from Node 2 and matches the received responseNonce with the saved nonce. Again, on a mismatch the message is discarded immediately. If the two nonces match Node 1 sets the SA to trusted. Finally both entities have performed their authentication and the next message received is passed to the IEEE 1588v2 stack. All messages exchanged in this procedure have to pass the ICV calculation, additionally timers are setup and if timeouts occur the connections are deleted without completing the whole procedure.

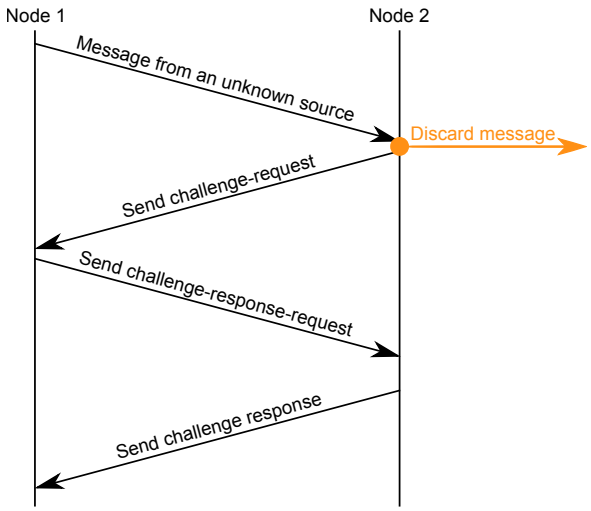

Figure 2.7: Chronological overview of the challenge-response procedure

Updating Security Associations

When the replay counter of a Security Association (SA) rolls over, an update procedure is triggered. The currently used key ID and lifetime ID are removed and replaced by next key ID and next lifetime ID. The update TLV delivers the new next key ID and new next lifetime ID, which is

going to be used for the current connection. The currently used lifetime ID is overwritten by the value of the next lifetime ID. Therefore, the next lifetime ID is empty and a new random value is generated to fill it. The new next lifetime ID has to obey two conditions:

1. It must not be zero.
2. It must be different from the current lifetime ID.

This update TLV has to be approved with a challenge-response authentication. This procedure provides new values for the next rollover and keeps the SA up to date.

During the communication with other entities, which have already established a Security Association (SA), the SA is used regularly. Such a periodic traffic is important to keep the SA in trusted state. If a SA is not used for a specified time, the connection times out and it is removed from the table. When a node wants to communicate with the entity again, the challenge-response procedure has to be repeated. The corresponding SA is already deleted from the SA table and therefore a new authentication of the node is necessary. This procedure is described in subsection 2.3.2.

The maintenance of established connections is implemented via security update TLVs, the structure of this TLV is depicted in Figure 2.7, it shows the structure of the TLV and the length of each field in byte. The *tlvType* identifies the type of the TLV, in this case SECURITY_ASSOCIATION _UPDATE. It is followed by the *length* field, which holds the length of the TLV minus 4 byte (*tlvType* plus *length*). The next field is the *addressType,* which identifies the connection type. Valid values are: All, Multicast, P-Multicast, and Unicast; The *nextKeyID* and *nextLifetimeID* are the values which replace the old *nextKeyID* and *nextLifetimeID* stored with the SA .

Type	Length
tlvType = SECURITY_ASSOCIATION_UPDATE	16
length	16
addressType	8
reserved	8
nextKeyID	16
nextLifetimeID	16

Table 2.7: Structure of an IEEE 1588v2 security association update TLV, with type and length (in byte) of the fields.

2.3.3 Internet Protocol Security

The Internet Protocol Security (IPsec) [KS05, Bri10] is a collection of protocols to secure Internet Protocol (IP) [DH98] communication. It includes protocols for key exchange, e.g., Internet Key Exchange (IKEv2 [KHNE10]), Internet Security Association and Key Management Protocol (ISAKMP) [MSST98], Kerberized Internet Negotiations of Keys (KINK) [SKTV06].

However, it also includes different services like the Authentication Header (AH) [Ken05a] or Encapsulated Security Payload (ESP) [Ken05b] and modes like Transport Mode and Tunnel Mode. These services and modes of IPsec are the subject of the following subsections.

Modes of the Internet Protocol Security

IP and IPsec are tightly integrated and both operate on the same Open Systems Interconnection (OSI) layer, on layer three. IPsec can handle both versions of IP, IPv4 and IPv6. The established connections are end-to-end connections between two IP nodes. IPsec supports two different modes of operation, transport mode and tunnel mode, explained subsequently.

Transport Mode

IPsec in transport mode authenticates and/or encrypts only the payload of the IP packet. The header of the IP packet is not modified nor encrypted. If the Authentication Header (AH) is used for transportation, it is not feasible to change a field, e.g., the IP address of the packet. That is due to the fact that the address is part of the hash value used by IPsec AH.

Due to the fact that all higher layers are protected by IPsec too, it is not viable to modify any bit of the data, e.g., translate port numbers. The Encapsulated Security Payload (ESP) is responsible for confidentiality. Figure 2.8 shows examples of IPsec packets in transport mode. Figure 2.8a and 2.8b depict the position of the newly inserted IPsec header after the original IP header for transport mode.

Tunnel Mode

IPsec in tunnel mode authenticates and/or encrypts the entire IP packet. Such connections are used to build Virtual Private Networks (VPN). The original packet is encapsulated in a new IP packet with a new IP header. Within this mode Network Address Translation (NAT) traversal is possible and poses no problem for the transmission. Figure 2.8c and 2.8d depict the position of the newly inserted IP and IPsec header before the original IP header.

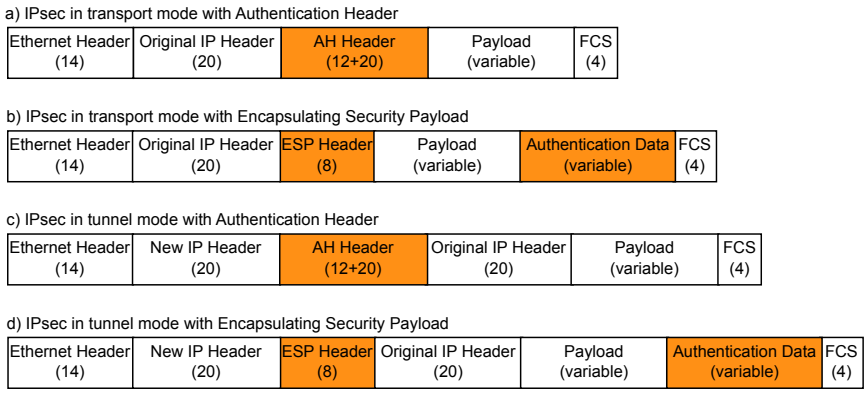

Figure 2.8: Combination of IPsec services and modes; additional parts in the packet introduced by IPsec are highlighted. The size of the different headers (in byte) is given in the parentheses.

Services of the Internet Protocol Security

IPsec offers two different services, Authentication Header (AH) and Encapsulating Security Payload (ESP). These two services can be used individually or combined. It is even possible to embed IPsec packets within IPsec, which offers the possibility to form complex network structures.

Authentication Header – Authentication and Integrity

The Authentication Header (AH) offers the functionality of connectionless integrity protection and authentication [Ken05a]. The authentication of the packet is determined by the Authentication Data (AD), which is the result of a cryptographic one-way hash function, called a Message Authentication Code (MAC). The hash is calculated over the complete IP packet including the header and is inserted at the end of the Authentication Header, which is depicted in Table 2.8.

Depending on the connection type, the calculation method used for the authentication can be divided into two cases:

- Point-to-point connections: Either symmetric encryption algorithms, e.g., AES and DES, or one-way message authentication codes, such as MD5, SHA-1, and, SHA-256 can be used.
- Multicast connections: One-way hash algorithms are combined with asymmetric signature algorithms.

Depending on the used algorithm, the length of the Authentication Data (AD) varies, e.g., 16 byte for MD5 [Riv92] or 20 byte for SHA-1 [Man07]. Excluded from the hash calculations are values, which will change or are not predictable during the network traversal, such as the Time To Live (TTL) field.

Table 2.8 shows the fields of an IPsec AH header. The packet starts with the *Next Header* field, which identifies the type of the payload. This value corresponds to the values used by IP. The *Payload Len* field specifies the length of the AH, in 32 bit words minus 2. To clarify this statement an example is provided. For the Table depicted in 2.8 the result of the calculation is a *Payload Len* of four. The header consists of 96 bit from the Authentication Header plus 96 bit for the Authentication Data, because a SHA-1 is used. The sum of 96 bit + 96 bit is 192 bit. The number of 32 bit words is 6, $\frac{192\,bit}{32\,bit}$. From this value "2" is subtracted resulting in a *Payload Len* of four.

The *Security Parameter Index* is an arbitrary number, which identifies the Security Association used by the connection. The *Sequence Number Field* is a counter, which is incremented with each use and is used as an anti-replay protection. At the end of the header, the *Authentication Data* is located, which comprises the result of the calculation of the integrity protection.

Type	
Next Header	8
Payload Len	8
Reserved	16
Security Parameters Index (SPI)	32
Sequence Number Field	32
Authentication Data (AD)	variable, 96 bit for SHA-1

Table 2.8: Authentication Header (AH) structure with type and length (in bit) of the fields

The composition of an IP packet with AH is depicted in Figure 2.8a and 2.8c. The AH header is located before the payload of the IP packet. Depending on the mode, which is used, the position of the AH header changes. For transport mode the AH header is located between the original header and the payload as shown in Figure 2.8a. In case of tunnel mode the AH header is located between the new added IP header and the original IP header, depicted in Figure 2.8c.

Encapsulating Security Payload – Encryption

The Encapsulating Security Payload (ESP) offers authenticity, integrity, and confidentiality for the IP packet [Ken05b]. ESP is designed for symmetric encryption algorithms and supports both block ciphers and stream ciphers [Man07].

When it comes to the offered services, authentication, integrity and replay protection, ESP takes the same approach as AH by including Authentication Data (AD) and a replay counter to the packet structure, but this data is appended at the end of the packet. Table 2.9 shows the structure of the ESP Header.

Similar to AH, ESP uses two different calculation schemes, depending on the application. A Message Authentication Code (MAC), based on symmetric encryption algorithms for point-to-point connections and one-way hash algorithms with asymmetric signature algorithms for multicast connections.

In ESP the length of the IPsec packet depends on two facts: First, the length of the Authentication Data (AD), which is determined by the algorithm, e.g., 16 byte for MD5 or 20 byte for SHA-1; Second, the length of the encrypted payload, which is determined by the block size[1] used by the algorithm, e.g., 16 byte for the Advanced Encryption Standard (AES) [DR02].

Table 2.9 shows the fields of an IPsec ESP header. The packet starts with the *Security Parameters Index,* which is an arbitrary number that identifies the Security Association used by the connection. It is followed by the *Sequence Number Field*, which is a counter incremented by one with each sending of a packet. This counter is used as anti-replay protection. The original IP packet is carried as payload in ESP. The *Pad Length* indicates the length of the padding, which is located in front of this field. Next to the last field the *Next Header* field identifies the type of the payload, this value corresponds to the values, which are used by IP. The *Authentication Data* is added as the last field of the packet and it comprises the result of the calculation of the integrity protection.

Type	Length
Security Parameters Index (SPI)	32
Sequence Number Field	32
Payload Data	variable
Padding	0-2040
Pad Length	8
Next Header	8
Authentication Data (AD)	variable

Table 2.9: Encapsulating Security Payload (ESP) structure with type and length (in bit) of the fields

The composition of an IP packet with ESP, is depicted in Figure 2.8b and 2.8d. It shows that the ESP is separated into a header, which is located at the beginning of the IP packet, and a trailer, which follows after the IP packet. The trailer comprises the Authentication Data (AD). Depending on the mode, the position of the ESP header changes. For transport mode the header is located between the original header and the payloada as depicted in Figure 2.8b. In case of tunnel mode the ESP header is located between the new IP header and the original IP header as depicted in Figure 2.8d.

[1] Block ciphers perform theit calculations on a fixed length of bits. The length of the stream of the bits is called the block size of the algorithm.

Replay Protection

Replay protection is implemented as monotonically increasing counter value, which is called Sequence Number in IPsec. This counter is increased by each use and is a mandatory field in the header of both services, the Authentication Header and the Encapsulated Security Payload. The sender always has to send this field, but it is up to the recipient if it is used or not. This service is only allowed to be used when authentication is used, otherwise the integrity of the value would not be guaranteed.

At the initialization of an SA the counter on both connection partners is initialized with zero. It is not allowed to produce an overflow on the Sequence Number. Before that happens a new SA connection has to be established and the counter is initialized again with zero. IPsec does not provide options to manage/synchronize the replay counter in a multi-sender environment, which uses a single SA. Therefore, the replay protection has to be deactivated in multi-sender environments using a single SA.

Network Configurations

With the help of the two modes and two different services provided by IPsec, different network connections can be established. Three generic connection types can be identified:

1. Network-to-network
2. Client-to-network
3. Client-to-client

These connections are depicted in Figure 2.9 and the explanations are following subsequently.

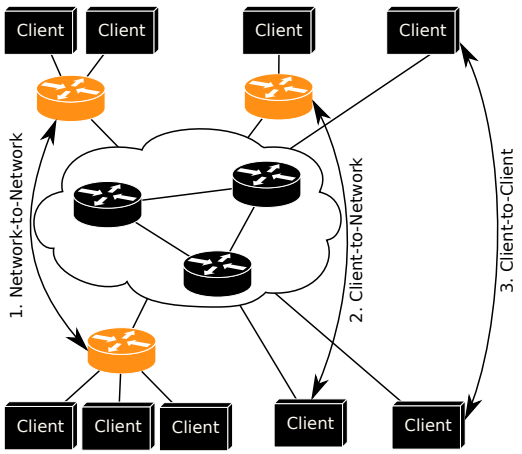

Figure 2.9: IPsec in three possible network configurations: network-to-network, client-to-network, and client-to-client

Network-to-Network Connection

A network-to-network connection is a connection which establishes a secure tunnel between two different networks. Between these networks any type of network configuration can be implemented. The two endpoints of the link act as border gateways for the two connected networks. The border gateways control the incoming and outgoing traffic of the two connection points.

The route of the traffic between the two endpoints is not predictable. This flow is depending entirely on the behavior of the intermediate network, which is not under the control of the border gateways. An IPsec connection is associated with a single Security Association (SA) and therefore only an end-to-end security can be established between the border nodes.

Clients connected to the tunnel in each network are not able to distinguish between traffic being sent or received locally and traffic being sent or received by the secure tunnel. An example for a network-to-network connection is depicted on the left of Figure 2.9.

Client-to-Network Connection

A client-to-network connection is a connection which establishes a secure tunnel between a network and a remote network device. Between the two members of the tunnel any type of network configuration can be implemented. The border gateway and the remote network device control the incoming and outgoing traffic of the two connection points.

Similar to the previously discussed network-to-network connection, the route of the traffic between the two endpoints is not predictable. The flow is depending entirely on the behavior of the intermediate network, which is not under control of the connected entities. Also in this configuration, the IPsec connection is associated with a single Security Association (SA) and therefore only an end-to-end connection can be established.

Clients connected to the border gateway of the tunnel are not able to distinguish between traffic being sent or received local and traffic being sent or received by the remote network device. An example for a client-to-network connection is depicted in the middle of Figure 2.9.

Client-to-Client Connection

A client-to-client connection is a connection establishing a secure tunnel between two remote network devices. Between the two members of the tunnel any type of network configuration can be implemented. Only the two endpoints can generate traffic and utilize the tunnel.

Similar to the previously discussed network-to-network connection, the route of the traffic between the two endpoints is not predictable. The path of the packet is depending entirely on the behavior of the intermediate network, which is not under the control of connected devices. Between the two network devices a single Security Association (SA) is established, an end-to-end connection.

The network devices know that traffic to the other network device can only be sent via the secured connection. An example for a client-to-client connection is depicted on the right of Figure 2.9.

3 Security Analysis for Clock Synchronization

During the last decades the information technology gained massive importance for industry and automation systems. This modern communication is very often the target of a variety of different type of attacks. Although the majority of threats are in the domain of computer science, this analysis concentrates on threats for network communication, it is the most relevant factor for clock synchronization.

3.1 Attack on Networks

The common application of clock synchronization is to provide this service for higher layers. Therefore an attacker who can manipulate the local clock of a node influences the functionality of the applications depending on it. Secure environments pursuit the following goals: confidentiality, integrity, availability, authentication, access control, and non-repudiation.

A classification of the attacks against clock synchronization networks can be categorized on the target that is attacked: master, slave, or the control loop [GTS06]. This classification should be extended for this thesis:

1. Master and slave: A first approach is to attack the nodes of the clock synchronization networks directly, e.g., master or slave. This attack heavily depends on the implementation of the stack used for the communication. In many cases it is valid to assume that the nodes themselves are secure. However, during this work a flaw in the specification was found, which is discussed in section 3.3. The worst case, apart from flawed code, is to block the communication of a node.

2. Selection algorithm: Modern clock synchronization protocols, e.g., IEEE 1588v2, use a simple approach to gather information about the other nodes. Every node announces the accuracy of the local clock. This announced accuracy cannot be checked by any of the communication partners. Things get worse when two nodes have the same accuracy and the decision is based on the MAC address. For IEEE 1588v2 the node with the lower MAC address wins the election. Spoofing the MAC address of a node is a well known method used in many attacks. When an attacker tricks the election it can setup a Byzantine master, also called babbling idiot master. This vicious master distributes wrong timing information and manipulates the part of the system relying on this service.

3. Network: A basic attack used to disturb the clock synchronization can be done by blocking packets traversing the network. This can be achieved either by physically interrupting the network connection or by deleting packets from the communication channel. Deleting packets can be performed in already installed network devices, e.g., switches or routers. A Denial of Service (DoS) attack directed against a switch or router results in a complete interruption of the network communication. A more sophisticated attack is to remove specific clock synchronization messages. It is much harder to detect and can influence the control loop of the nodes. The manipulation of the control loop affects all kinds of messages used for communication.

Apart from these specialized attacks for clock synchronization, a more general definition can be used. An attack is initiated by an adversary who can be classified as follows [Sch95]:

- Passive attacker: This type of attacker only observes the traffic exchanged over the network. The attacker is only a threat for the confidentiality of the data.
- Active attacker: The adversary is working actively to gain influence. This might be achieved by deleting messages from the network, adding traffic, altering information, or by simply delaying particular messages. In this case the attacker compromises the integrity, the authentication, availability, and also the confidentiality of the data.

According to this classification, attacks on clock synchronization systems are:

1. Eavesdropping: It characterizes the behavior of an adversary who is listening to private conversations of third parties without permission. In the case of network communication, the adversary monitors the network traffic. If data is sent without encryption, the transmission is cleartext and the attacker can read the traffic. As already outlined in subsection 2.3.1 and subsection 2.3.2 confidentiality is of minor interest for clock synchronization networks, because the time is seen as a common good.

2. Man-in-the-Middle: The attacker is located between two communication partners and impersonates their identity. To successfully establish a connection, the adversary fakes the identity of the first entity to communicate with the second entity and fakes the second entity to establish the communication with the first entity.

3. Data Modification: The attacker gains access to a network and modifies transmitted data. The influence does not stop with modifying data – if an attacker can modify the traffic, it can also redirect, delete, or generate artificial traffic. This is exceptionally easy when the transmissions are cleartext without any protection, such as hashes or encryption.

4. Replay Attack: The adversary observes the traffic and saves valid transmissions for later use. At any unspecified time the packets are sent again. For the domain of clock synchronization packets used to synchronize a clients clock are replayed at a later time, thus the client adjusts the clock to an "old" time. Services depending on the correct time cannot fulfill their task anymore.

5. Denial of Service (DoS): A Denial of Service attack has only one goal, to shut down network communication. To bring down a connection the interface of the victim has to be overloaded. The goal can be achieved when a lot of messages are sent to the victim. It becomes impossible

to process all packets and the system is overloaded. A variant of this attack is the Distributed Denial of Service (DDoS) attack. Opossed to DoS, DDoS uses a distributed approach and sends the messages from different points in the network to overload a common target.

6. Delay Attack: As the name suggests, this attack delays specific messages, which results in a negative impact on the network. For the domain of clock synchronization where time is the most critical asset, a synchronizing node is extremely vulnerable to these kind of attacks. Messages, which are used to determine the delay between a server and the client, can be delayed. This is resulting in a wrong value for the network traversal time compensation, which has direct influence on the adjustment of the clock.

7. Byzantine Clocks: This type of attack is specific for clock synchronization. The byzantine clock is a type of clock, which does not simply fail, instead it produces an output, which leads to a maximum error within the network. If applied to the domain of clock synchronization, a byzantine clock would provide timestamps, which produce a maximum time difference between the synchronized components.

3.2 Inefficiency at the Startup of Secure PTP Networks

In the PTP network without security extension, when a node is started it waits until an internal timeout occurs and then starts sending out a synchronization message. All the other nodes in the network perform the same procedure and send a synchronization message. With this mechanism each node collects data from the other members in the network and then performs the Best Master Clock (BMC) algorithm to determine the master. Each node sends one message and all other participants in the network have the information needed.

The startup of a network with security enabled looks different: When a synchronization message is received by a node it performs a lookup in the Security Association table. As the node just got started (and the table is empty) the search returns with an empty result for the lookup. When a sender is not known, the message is discarded immediately and a challenge-response procedure is triggered. This behavior can be assumed as a standard procedure for newly started nodes. All nodes in the network follow the same rules and all initially sent messages are discarded, shortly followed by the messages generated by the challenge-response procedure, resulting in a flood of messages during startup. After the nodes have authenticated each other, another timeout has to occur before sending the synchronization message. This time the BMC algorithm can be performed and a master is selected.

To eliminate the flood of PTP packets, at the beginning two actions can be performed:

1. The initial synchronization message is not discarded but re-evaluated after the authentication of the nodes. This action reduces the time needed to elect the best master. The additional timeout, which is needed to perform the BMC, can be saved.

2. The timeout until the first synchronization is sent can be adjusted. It is possible to define a minimum and a maximum value via a configuration switch for the startup phase. A random variable is used to determine the timeout positioned somewhere between the two boundaries. This action does not reduce the overall bandwidth, but flattens the load.

3.3 PTP Security Flaw

A typically clock synchronization network consists of several network components:

- One master clock, which distributes the time to all other nodes in the network.
- Slave clocks that synchronize to the master of the network.
- Transparent clocks, which are actually switches with additional timing capability. They measure the time a PTP message needs to pass from the input port to the output port. This is necessary, as the switching fabrics in the switches have a non-deterministic behavior, hence the transparent clock compensates this drawback.

These components together with the connection are depicted in Figure 3.1. Additionally two adversaries are inserted. The first, on the left side of the figure, is a man-in-the-middle attacker. It is located between the master and a slave. The second attacker is a transparent clock, which can reach a higher number of nodes in the clock synchronization network.

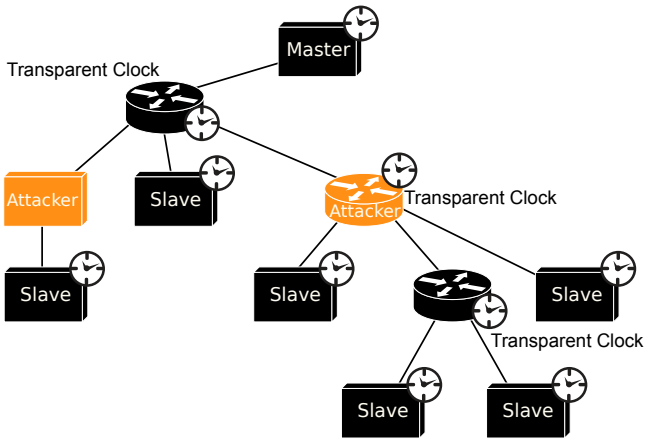

Figure 3.1: Structure of an IEEE 1588v2 network with attackers

3.3.1 Modification of Source Address

For optimized transmission, the size of the PTP messages is kept at minimum. For the sake of efficiency, the security extension reuses the network protocol address. Incoming messages are verified with their respective Security Association. The Security Association is identified by the source port ID, source protocol address, destination port ID, and destination protocol address. From a security point of view this behavior does not affect the security of the system. However, together with the scheme used for the ICV calculation a major security breach is produced.

The IEEE 1588v2 standard specifies:

> "Using the message authentication code function selected by the algorithm and secret key, the ICV value is computed over all PTP message fields beginning with the first octet of the common header and ending with and including the last octet of the security extension TLV. " [IEE08, page 244]

This specification results in the breach of security [TH09]. The network addresses, which are used as identifiers, are not protected by the Integrity Check Value (ICV) and can be changed by an attack without being detected. Due to this security breach the possible attacks can range from a Denial of Service (DoS) attack, by filling up the Security Association table, triggering unlimited challenge-response cycles, or even a replay attack with old synchronization messages.

3.3.2 Creating Counterfeit Security Association

A Security Association consists of several parameters: source port, source protocol address, destination port, destination protocol address, key, random lifetime ID, and a replay counter. These informations are stored in a SA table, with the four values of source and destination address forming the unique key.

The basic principle needed for this attack, is to create a new Security Association from an already existing one, or to clear an already established Security Association. For this attack, a villain is positioned between a master and a slave, as shown in Figure 3.1 path over device (a) or (b). The attacker has the ability to change and replay packets. According to the specification a master and a slave perform a mutual authentication at the beginning of the communication. The procedure results in trusted Security Associations on both nodes. The complete communication cycle, together with the information stored at the nodes is depicted in Figure 3.2. At the end of the mutual authentication each node has authenticated the counterpart and has created an entry in the SA table with all relevant information.

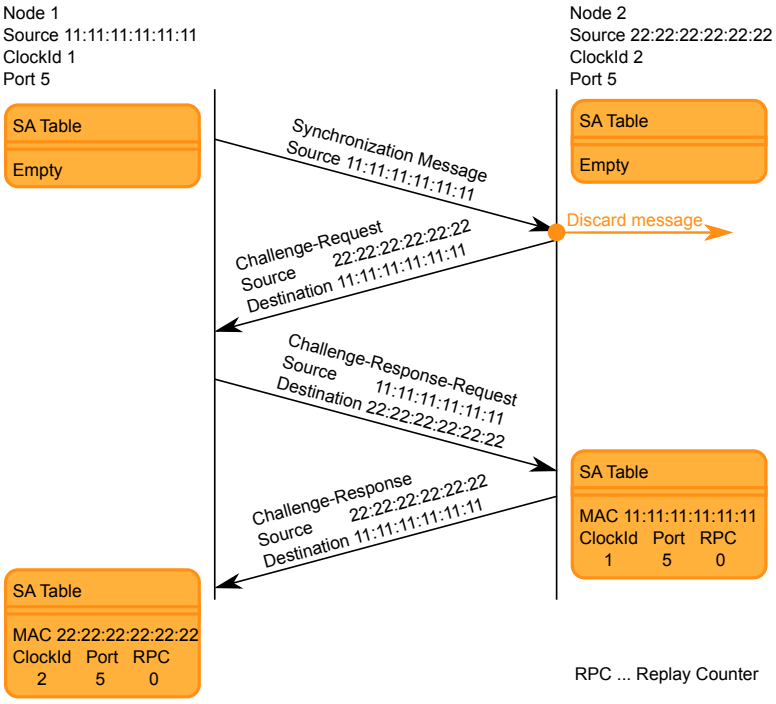

Figure 3.2: Mutual authentication procedure

A successful attack starts with modifying the MAC address of a regular message, e.g., synchronization message. The modified MAC address is the address of the attacker, AA:AA:AA:AA:AA:AA. The altered message is replayed by the attacker and triggers a new challenge-response procedure.

Due to protocol address (not part of the ICV), the ICV of the message is valid and passes the ICV check at the receiver. This message is then processed as a legit query and a normal challenge procedure is performed. In Figure 3.3 the process of creating a counterfeit SA entry is depicted. The attacker replaces the original MAC addresses 11:11:11:11:11:11 and 22:22:22:22:22:22 with its own MAC address, AA:AA:AA:AA:AA:AA. The result of this procedure is a new trusted SA entry with a fresh initialized replay counter. Such new SAs can be created as long as the limit for dynamic SAs is not reached.

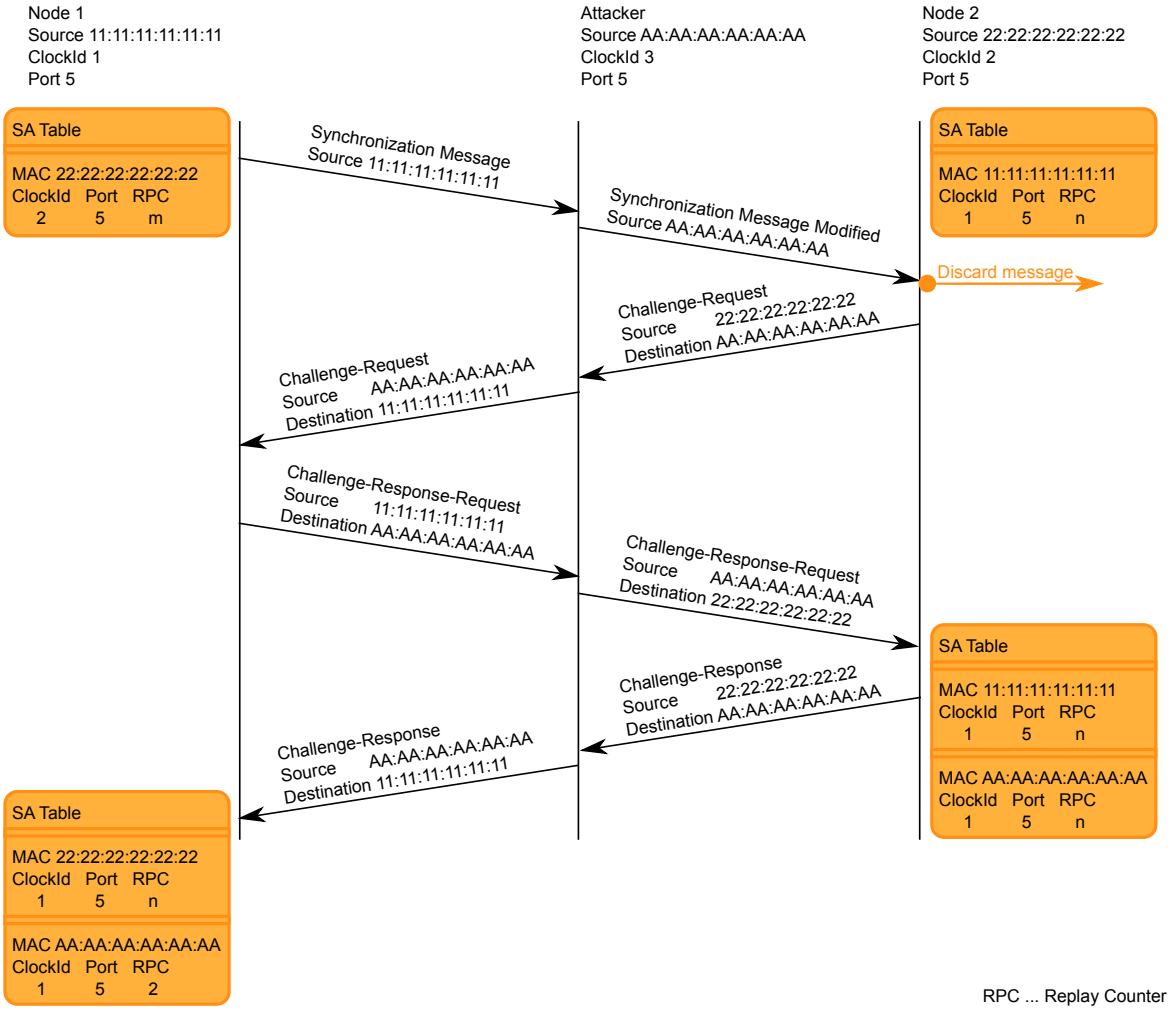

Figure 3.3: Vicious mutual authentication procedure

This attack can be used directly to flood the Security Association table of the attacked node with counterfeit, although fully trusted entries. This behavior offers the possibility to launch Denial of Service (DoS) attacks against both types of nodes, master and slave, and prevent legitimate connection attempts.

When the attacker forces already established Security Associations into timeouts, the complete Security Association table can be filled up with these counterfeit SAs. The attacker can accomplish this task with low effort: he removes messages completely or modifies them a bit to prevent a positive ICV check. This attack does not only fill the Security Association table, but as the generation of these entries only differ in the MAC address, recorded packets can be replayed.

The element which softens the effect of the attack is the lifetime ID. This is a 16 bit integer value that is set randomly and during the attack the value for the counterfeit SA entry is not changed. Therefore the success of the attack depends on the implementation of the random number generator and the frequency of the attacks. The frequency of attacks is limited by the creation of new SAs, which depends on the speed of the challenge-response procedure.

3.3.3 Resetting Security Association by Timeout

This attack shows a second alternative to reset the replay counter of an attacked node. Similarly to the attack used in subsection 3.3.2, an authentication between two nodes is performed, resulting in a valid Security Association (see previous Figure 3.2). In this case the attacker blocks the traffic between the master and a slave. Thus the Security Associations are forced into timeouts and a dynamic SA is deleted or a static SA wil expire. Subsequently, the attacker opens the communication channel again and the authentication is performed once again. After this re-authentication the attacker has two possibilities:

1. In the case of a dynamic SA, a new dynamic SA is created. The only issue left for the attacker is to match the lifetime ID, similar to the attack described in the previous subsection.
2. For static SAs, no particular specifications are given in IEEE 1588v2 how to proceed after such a connection has expired, only that values of such connections have to be preset. Thus, the attack depends on the implementation of the stack. For implementations, exactly following the standard and presets all the values, it can be assumed that the lifetime ID is also affected. Therefore, an attack is possible.

3.3.4 Replay Attack with Counterfeit Security Association

A replay attack starts with recording the traffic between a master and a related slave. The packets can be recorded anytime, it is not necessary that the counterfeit Security Association is already set up. The prerequisites for this attack to work out are:

1. A reset Security Association in the counterfeit SA (see section 3.3.2 and section 3.3.3).
2. A matching lifetime ID.

Under these two conditions the recorded packets of the attacker can be used for the attack. The attacker replaces the original source protocol address with the source protocol address of the counterfeit Security Association (see section 3.3.2). After that, the packet can be replayed by the attacker. The modified packet looks legitimate for the receiver, because the ICV calculation is performed correctly. The last issue, which could prevent the attack, is the sequence ID, which is used by the PTP stack. However, the security extension is passed and the packet is forwarded to the clock synchronization stack. This way the attacker has replayed a clock synchronization packet successfully.

Although the message has passed the security extension, the sequence ID of the PTP packet could hinder or delay a successful attack. The sequence ID of the PTP packet has to be incremented by one on each packet received. For each type of message and destination address a separate

sequence ID counter has to be used. No rules are specified for verifying a received sequence ID. An implementation may check that the counter is increasing or use a more stringent rule, the counter has increased by one. The stringent rule would not tolerate lost messages, which can be a big problem in real life scenarios. The worst case scenario for an attacker happens when the attack has to be delayed until the sequence counter rolls over and the messages can be used again. An attacker which removes all messages from the communication path can change the value of the slave clock according to the recorded packets, only limited to the messages with the current lifetime ID.

This attack uses the fact that a 2 byte sequence ID of the PTP stack has a higher frequency of rollovers than the 4 byte replay counter of the security extension.

3.3.5 Replay Attack with Static Security Association

Similar to the previous attack, the traffic between the master and the corresponding slave is recorded. After recording the communication, the static Security Association is reset (see section 3.3.3). The attacker can start sending the recorded packets and at the same time blocking the communication from the master. Following the previous attack, only those messages can be replayed, which have been recorded at the beginning of the attack. The maximum amount of messages which can be facilitated depends on the replay counter. The replay counter is 32 bit long, hence a maximum number of 4294967296 (2^{32}) messages can be recorded. After the rollover of the replay counter the lifetime ID is changed, which deprecates the recorded messages.

However, when a static Security Association is reset by the attacker, the recorded messages can be used multiple times. The only drawback of this approach is again the sequence ID. It is required to be larger than the sequence ID of the previous message. Due to these implementation specifics, the attack might have limited success when the check of the sequence ID is more stringent.

3.3.6 Effect of the Vulnerability

The vulnerabilities, found during the analysis of the IEEE 1588v2 standard, affect both master and slave. Furthermore, the effect is not only limited to Ethernet, any transport protocol used to transport the PTP messages allows replaying messages. The clock value can be changed, as pointed out in the previous subsections. The effect is not limited to synchronization messages, it also affects other messages, e.g., management messages, which can be used to alter the configuration of a node.

When these modified messages are passed on by the security extension, higher functionalities such as master clusters, are also affected. To be still compliant with the PTP standard, the only option left is to ban the use of dynamic Security Associations and use only static Security Associations, where the protocol address is predetermined. With this concept the creation of counterfeit Security Associations is avoided and the effect on incoming SAs is limited. These types of connections have to be limited to only one connection, otherwise the connections can be abused when a counterfeit Security Association is created.

The attack itself has to be rated high. The security extension uses unprotected data to authenticate a connection. As shown in the previous subsections, several kinds of attacks benefit from this behavior. However, the attacks rely on implementation specifics of a stack and additionally have to use brute force attacks against the lifetime ID. Furthermore, only a receiver of the message

is affected. Therefore, transparent clocks relaying messages, are not directly affected, only if the destination of the message is the transparent clock itself.

4 Design and Implementation of a Secure Clock Synchronization Stack

This chapter puts a focus on the design and implementation of a secure clock using the IEEE 1588v2 Annex K security extension. The approach is derived from the design goals, which are outlined in subsection 4.1. It is followed by a description of the implementation of the security extension in subsection 4.2. After the solution provided by Annex K, a second approach using IPsec to secure the PTP traffic is presented in subsection 4.3.

4.1 Design Goals

The task of extending and, in this case, also securing an already existing system, has to be carried out carefully. Therefore, the task of planning and integrating these new parts is of utterly importance.

Software design is the procedure of solving a given problem and implementing the solution in some kind of programming language. The whole process of designing a new software starts with a need that arises and has to be satisfied. The design process includes the task of defining an software architecture, implementing algorithms and also taking into account the hardware, which has to be used efficiently.

The so called needs that come up can be manifold. For a system, which is used in a time resource limited environment, efficiency might be the factor with the highest impact. Another system has to work in very demandig environments with a lot of errors. The demand here would be to stay in a safe state. The simplest case could be that a task is periodic in nature and the procedure of solving it manually is not very hard and only costs time. Another possibility is that a task is too complex and time sensitive so that it is impossible for a human being to fulfill this need. Either way the process of designing an optimal solution for a given problem is a challenge. In the course of this work the need is to extend an already available PTP stack with a security extension.

The first step is to perform a thorough software requirements analysis. This process is the foundation for the specifications, which are needed to engineer the software. The specifications are already available in the form of an IEEE standard, namely IEEE 1588v2. The functional requirements of security for the different use cases are specified in Annex K of IEEE 1588v2.

For this work, a PTP stack conformant with the specifications of IEEE 1588v2 has been provided by Oregano Systems and is available for implementing the security extension. The provided stack is platform independent and runs under Windows and Linux.

To be able to support both operating systems without having to maintain two versions, the security part has to be located after the platform dependent functions. For security reason, it is favorable to take a modular approach. This ensures that the new functionality is encapsulated, has fixed points for interaction with the rest of the system and helps on maintenance [TH08]. An implementation, which is also transparent for the stack, allows an almost independent development of the security and the clock synchronization stack. Furthermore, the application domain dictates an efficient implementation: The clock synchronization stack and its security extension are used on resource limited devices, and are deployed in a network that uses the timestamp information as foundation for other services. As a conclusion, three main attributes for this integration are:

1. Modular layout
2. Transparent design
3. Efficient implementation

After the implementation phase (integrating all functionalities), the new software needs to be tested. This procedure ensures that the new software works as it supposed to be. The test procedure is divided into two parts, verification and validation. Verification is about building the system right and validation ensures building the right system.

In the following sections the integration, the verification, and the validation of Annex K into the the provided stack is explained.

4.2 Integration of Security in the PTPv2 Stack

As pointed out in the previous section, the design has to favor a modular, transparent and efficient implementation. For obvious reasons, the security extension may not mess with the core functionality of the PTP stack and has to keep the overhead low. Furthermore, an inspection of the code and the interfaces from the code has to be performed to avoid the possibilities of side-channel attacks, buffer overflows and other attack patterns.

The security extension is implemented as an extra layer, which incorporates the security extension. The architecture is depicted in Figure 4.1. The advantage of this architecture is the highly modular structure as defined as essential for this application. This approach offers the possibility to expand the security extension independently of the core functionality. Additionally, it is possible to modify certain aspects of the extension to test different implementations for certain parts, e.g., different cryptographic libraries or variable back ends for data handling.

The next module, located at the bottom of Figure 4.1 is the PTP network interface, named *PTP_NetworkIfc*. It is part of the provided PTP stack, has to split the received messages. This fragmentation divides the head of a PTP message from the appended Type Length Values (TLVs). This module is not part of the security layer, however this module has to work very efficiently to ease possible Denial of Service (DoS) attacks and to prevent buffer overflow attacks. PTP_SecurityIfc is the interface class between the PTP_Engine and the security extension. The

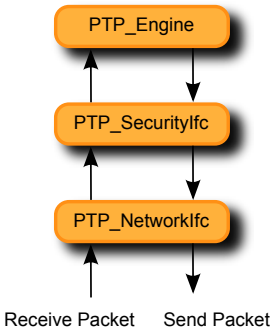

Figure 4.1: PTP stack architecture, with added security layer, *PTP_SecurityIfc (Ifc...Interface)*

security module has two interfaces, one to the upper module, the PTP state machine, and one to the lower module, the network interfaces of the system. The security extension has control over the network traffic, incoming and outgoing, which is destined for the PTP stack. It performs all security affiliated tasks between the upper and the lower layer modules. Finally the module *PTP_Engine* represents the PTP stack.

The communication with the PTP state machine has to be divided into a receiving and a sending part. When a message is received, the security extension decides if the message is passed on to the state machine. This decision is provided via an integer value, which indicates if the message has to be discarded or is handled as usual. For a simple case, a boolean would be enough as return parameter, however for advanced feature implementation an integer is used. Not only enables it the stack to discard the message, it also can trigger an indicator for a specific security breach. The sending of a message is also transparent for the PTP state machine. The return value of the sending function responds with sending successful or sending failed, exactly to the feedback provided by the network module directly. Therefore the PTP stack gets a feedback, which is the same for activated and deactivated security extension.

The base of the security module builds on the security association table. This table contains all the information necessary for running the security module and also supplies functions to add, remove and search for entries. The security extension Annex K specified in IEEE 1588v2 demands that all security relevant data of the message have to be checked against the locally stored data. The same applies for sending a packet, which also results in retrieving data from the security association table to compose a valid message for the receiver. Sending and receiving PTP messages is a periodic task. In particular for a master clock, which supplies several hundreds of slaves, the computational load can grow very fast. Therefore, the operations on the security association table are optimized for fast information access, which use well performing algorithms.

C++ offers different types of lists, e.g., hash map, linked list, double linked list, etc. The security association table makes use of hash maps, which are included in the standard template library and offer good performance. The main class of the security module calls the appropriate functions to perform the relevant tasks such as adding and deleting of new Security Associations. The security state machine takes care of keeping track of the various messages and timers which are due. The expiration of authentications, expiration of security association, and the update of security associations are also handled by the security state machine.

4.2.1 Structure of the Security Module

The modular approach for the security class, which is explained in the previous chapter, is used to segment the implementation into specific parts. The implementation makes use of already implemented types, classes, and functions of the PTP stack, which keeps the overall size of the stack slim.

The structure of the security module is depicted in Figure 4.2. On the left side of the diagram the security class itself is shown. This class is only allowed to be instanced once. It supplies all functionality within this single module and can handle hundreds of connections. It is only limited by the storage space and the computing power of the supplied system.

The security module depends on several classes, which can be seen on the right side of the diagram. To be able to instance the *PTP_ Security* class, a *PTP_ LogIfcError* has to be supplied for logging together with a *PTP_ ConfigIfc* to retreive configuration parameters.

1. *PTP_ LogIfcError*, this class is the back end for all logging tasks. It provides several logging levels, i.e., *Error*, *Warning*, *Notice*, *Info*, and *Debug*. These levels provide different verbosities for the user. The most verbose level is *Debug*, used for developers to get the maximum amount of information from the stack, to track down errors and malfunction and to find the part of the code responsible for these actions. The next higher level is *Info*. It supplies less information than the *Debug* level, but enough output for maintenance to see if severe problems occured. The next level is *Warning*, supplying a user with warnings of the stack, which might cause problems during runtime. The last level is the *Error* mode, which only reports severe problems to the user.
2. *PTP_ ConfigIfc* supplies the stack and the security module with configuration settings. For the security module, these settings concentrate on the keys and associated data. This data is read from a configuration file. The file uses the a Comma Separated Values (CSV) structure where each dataset starts with a new line.

The format of the *PTP_ ConfigIfc* follows this layout:

- An unique key ID, to identify the key.
- An algorithm ID, which specifies the algorithm that has to be used. Two values are possible “1” for SHA-1 or “2” for SHA-256
- The key, which is used to ensure the integrity of the data.
- An expiration time, which indicates the expiration of the validity of a key.

Both classes are supplied via the PTP stack and are also used by the stack itself. The Type Length Values (TLVs), needed by the security extensions, are inherited from the base classes supplied by the PTP stack via the *PTP_ TLVBase* class. These three types are:

1. PTP_TLVAuthentication
2. PTP_TLVAuthenticationChallenge
3. PTP_TLVSecurityAssociationUpdate

Figure 4.2: Overview of the security module class structure

The relation of the classes is depicted in Figure 4.3. Each of these message classes represents the structure of the message type it belongs to. Furthermore, two functions are available:

- Network-to-host byte order
- Host-to-network byte order

The two functions are needed to support a platform independent approach. Different types of computers are using different byte orderings, big-endian (most significant byte first byte in the word, default for network byte order) and little-endian (most significant byte last in the word, default for x86 CPU from Intel and AMD). With a conversion between the two schemes, communication systems with different byte orders can communicate with each other.

The next three subsections will give more detail on the main actions of the stack sending a message, receiving a message, and the challenge-response procedure.

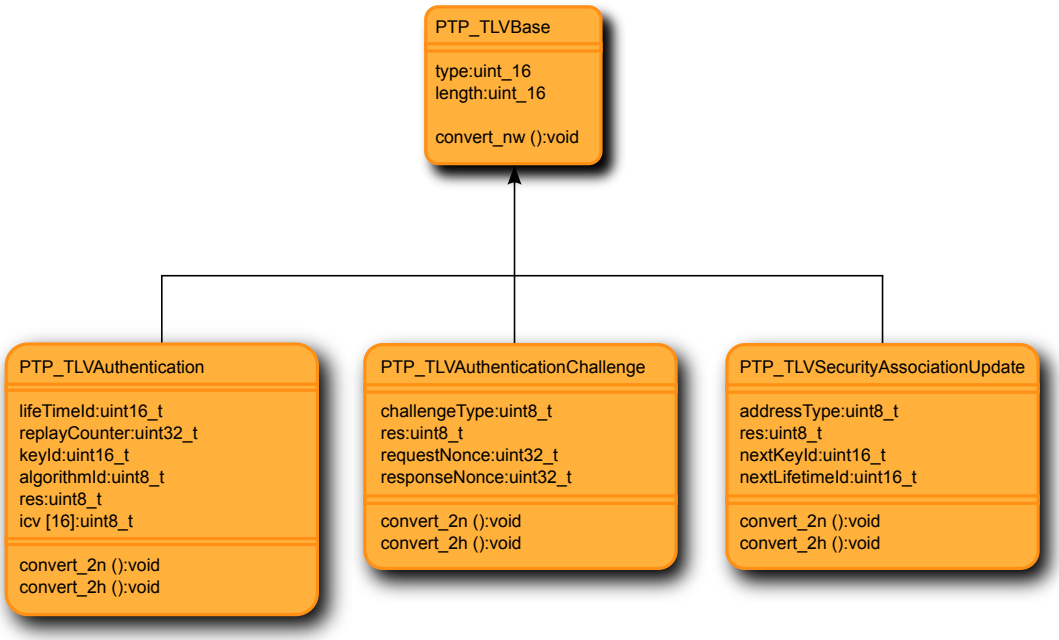

Figure 4.3: Message types specified by Annex K in IEEE 1588v2

Processing Incoming PTP Messages

The processing of an incoming PTP messages is depicted in Figure 4.4. The PTP stack calls the security extension and passes on the received message. In the first place, a review of the message to validate it and build the base for further processing is performed:

1. Check if the security flag is set in the header of the PTP message, see section 2.2.2.
2. Check if the PTP message has an appropriate Authentication TLV appended.
3. Check the ICV. The module "Message is valid" in Figure 4.4 performs this first check.

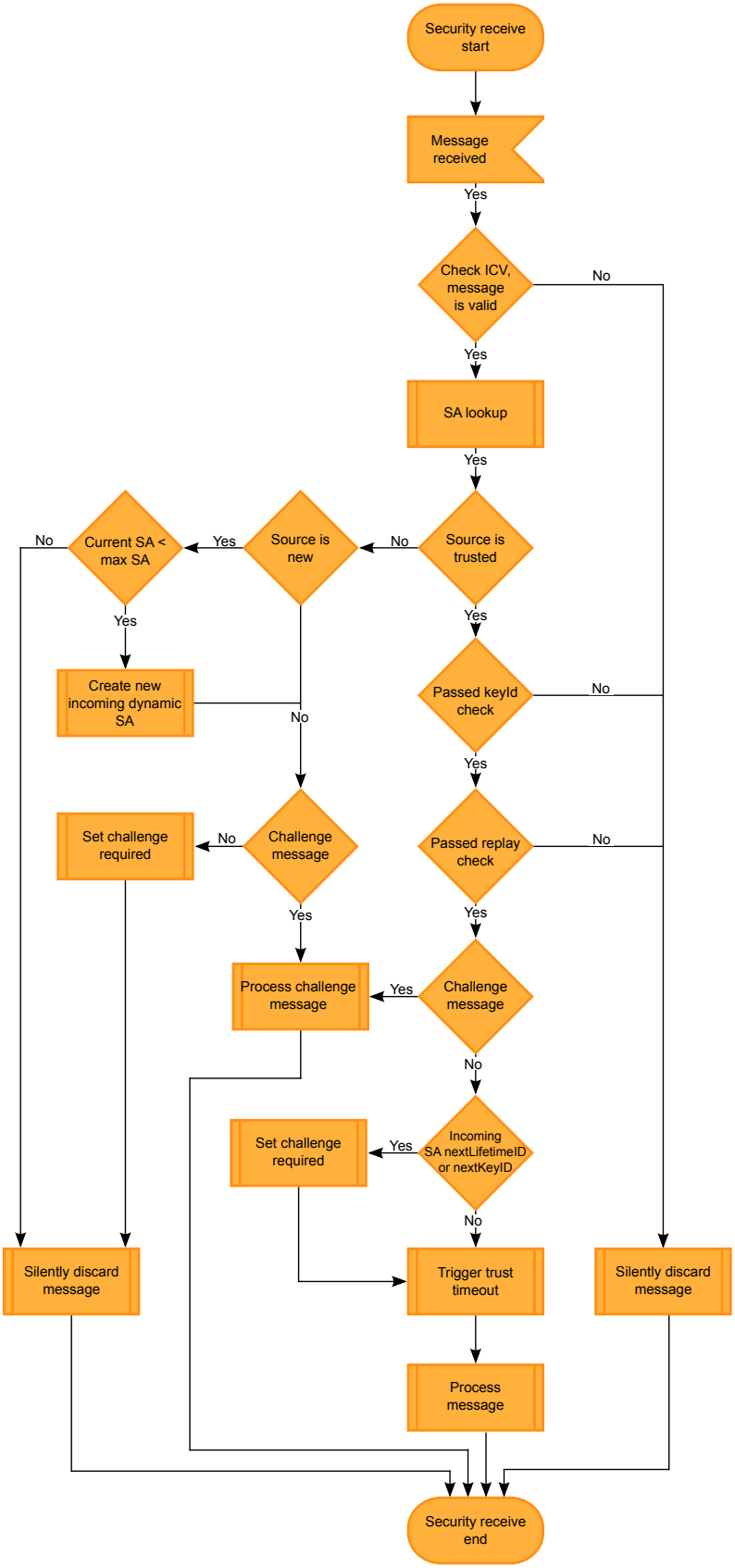

Figure 4.4: SDL diagram of the secure message processing of the IEEE 1588v2 version two security extension

If one of the validation fails, the message is immediately discarded. This behavior helps minimizing the impact on the system load. After this entry-level test, the message is passed on to the state machine of the security module.

The security state machine is performing a look-up in the Security Association Table to find a Security Association (SA) related to the incoming message. Depending on the outcome of the search, two different options are possible:

1. The look-up brings up an empty result. There are currently no associations built up with the sender of this message. If the number of currently established associations is lower than the maximum associations allowed by the system, a new association is created together with the initial challenge-response message. All new and updated associations have to be verified via challenge-response message exchange.
2. There is a, SA available for the message. The entry in the Security Association Table provides additional parameters, which are checked against the received message. The stored key ID in the message has to be the same key ID, which is saved with the corresponding SA and the replay counter has to be greater than the stored one.

In the second case the next steps depend on the message type, which is received.

- If the received message is a challenge-response message, it is processed according to section 4.2.1.
- The message uses the next key ID and next lifetime ID. These parameters are also part of the Security Association Table and can be checked. The switch from old security parameters to new security parameters has to be confirmed by a challenge-response procedure.
- If the message does not fit into one of the previous two options, the timeout of the SA entry is renewed and the message passed on to the PTP stack for further processing.

Challenge Processing

The challenge-response processing is a special case of receiving a message. Figure 4.5 shows the functions in detail, which are presented as block "Process challenge message" in Figure 4.4.

When an incoming message is identified as a challenge-request, the security state machine prepares a challenge-response-request message. The response-request message includes a request nonce, which has to be saved to validate a proper response.

If a challenge-response or challenge-response Request is received, a valid association has to be present in the Security Association Table and the status has to be challenging. If there is no entry or the status is not challenging, then there is no pending request and, therefore, such a response is bogus and the message is silently discarded.

For the case that a Security Association is found, which is in state challenging, the nonce which is part of the message (see section 4.2.1) is checked against the value stored in the Security Association Table.

If the message of type response-request is a matching response, message is compiled and sent back as an answer.

After the message has passed these checks, it is determined that this message is an adequate response to a request, which is expected. Therefore, the state machine modifies the Security Association and updates it with the data delivered in the message. For a successful response-request or response, the following data is modified:

- The trust state is changed to trusted.
- The challenge state is changed to idle.
- The lifetime ID of the message is copied to the Security Association to prolong the lifetime of the Security Association.
- The replay counter is updated to reflect the actual count.

Send Message

The process of sending a PTP messages is depicted as flow diagram in Figure 4.6. The PTP stack calls the security extension and passes on the message, which needs to be sent. The need to facilitate the security module is determined by the security flag, which has to be set in the PTP header (see section 2.2.2).

All PTP messages to be secured are extended by the Authentication TLV. The data, which has to be filled in this TLV is provided by the Security Association. The following data is inserted:

- The current value of the replay counter.
- The matching lifetime ID for this Security Association.
- The key ID for the key currently in use.
- The algorithm ID, which corresponds to the key ID.
- The result of the ICV calculation.

The next step is to prepare the replay counter for the next sending process. The counter in the Security Association has to be increased by two after every use [IEE08, section K.2]. During this step the replay counter can roll over and an update procedure is triggered (see section 2.3.2). The last step in this update process is to set the replay counter to zero. When all these steps are completed, the message is compiled and can be sent.

4.2.2 Implementation Details

The security module uses several interfaces to interact with the adjacent modules. To give an overview of the implementation, the most important interfaces are introduced subsequently.

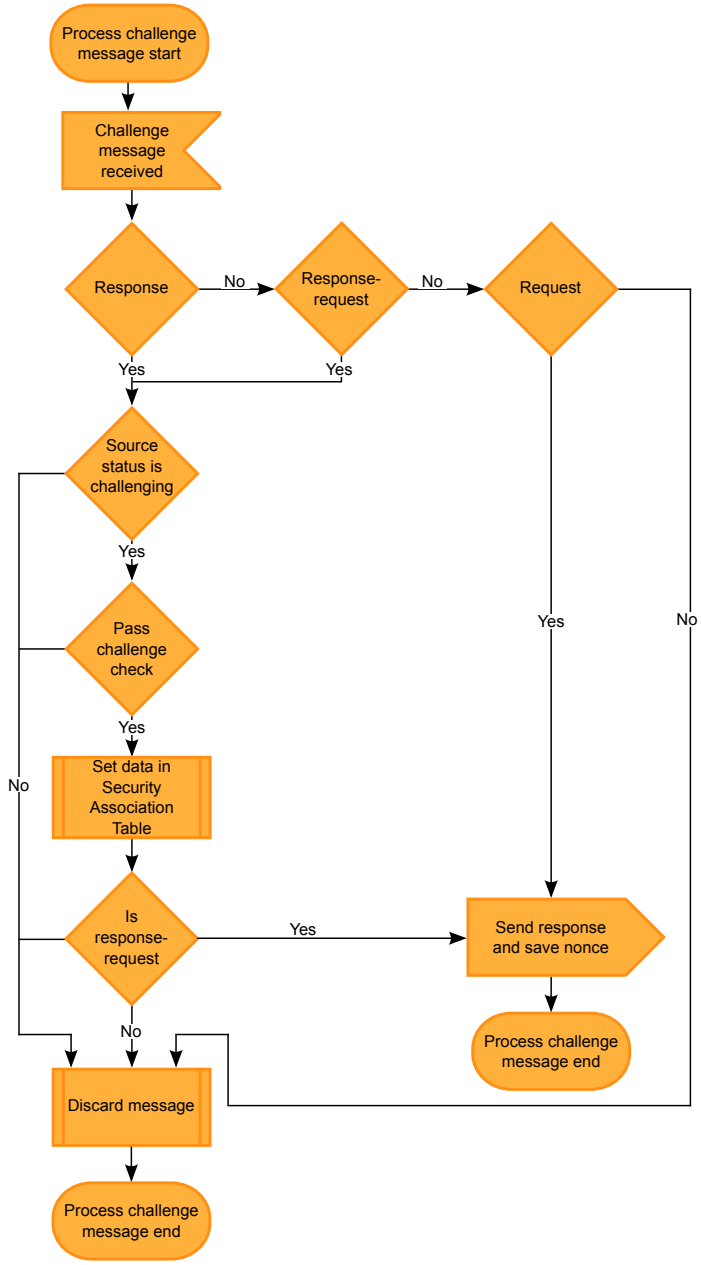

Figure 4.5: SDL diagram of the challenge-response routine from the IEEE 1588v2 security extension

Security Associations

The Security Associations defined by Annex K of IEEE 1588v2 are mapped to an internal structure in the security module. This internal structure is based on the information supplied in Annex K and is extended with additional data for the security state machine. Every SA consists of the following information:

```
struct saEntry {
 PTP_PortId portId;
 uint32_t replayCounter;
 uint16_t lifetimeId;
 uint16_t nextLifetimeId;
 uint16_t keyId;
 uint16_t nextKeyId;
 security_state trustState;
 timeval trustTimer;
 timeval trustTimeout;
 challenge_state challengeState;
 timeval challengeTimer;
 timeval challengeTimeout;
 uint32_t requestNonce;
 uint32_t responseNonce;
 sa_type saType;
 bool requestSent;
 bool responseRequestSent;
};
```

This SA structures are maintained in tables and are used in the incoming and outgoing Security Association Table. The key list uses a similar approach.

The tables, which are used in the implementation, have a big impact on the overall performance of the security module. Many of the operations performed in the security module depend on inserting, deleting, and searching for specific security associations.

There are a lot of implementations available for handling such lists, e.g., linked list, double linked list, circular linked list, hash list, etc. For this implementation there are several demands, which have to be fulfilled. The implemented list management has to cope with the available processing power and limited space. It is senseless to implement a database with all the functionality to handle the information occuring on a network node. Another need which arises, is the speed. A network node in a small network may handle only 10 nodes at the same time. However, in big networks, this number can grow to several hundred nodes.

An implementation fulfilling all the demands is the so called hash map, also known as hash table or unordered map in C++. Hash maps offer almost similar look-up costs independently of the number of elements stored in the list. The hash map uses a hash function to map values to known keys. In this implementation the port ID is used as key, because it is an unique identifier within the PTP network. The hash function transforms each key and builds an index for searching. In theory a hash function maps each possible key to a unique entry in the index. However, this ideal cannot be met for all situations, although this incident is a very rare case. In such exceptions, the implementation in C++ is providing handlers, which detect such incidents and are able to resolve them.

The hash maps are used for the incoming and the outgoing Security Association Tables and also for the list of keys. The implementation of the lists offers functions for inserting, deleting and searching within the list. To perform this task, additional functions are needed to traverse the list. To provide an example, the functions related to the incoming list are depicted in Table 4.1.

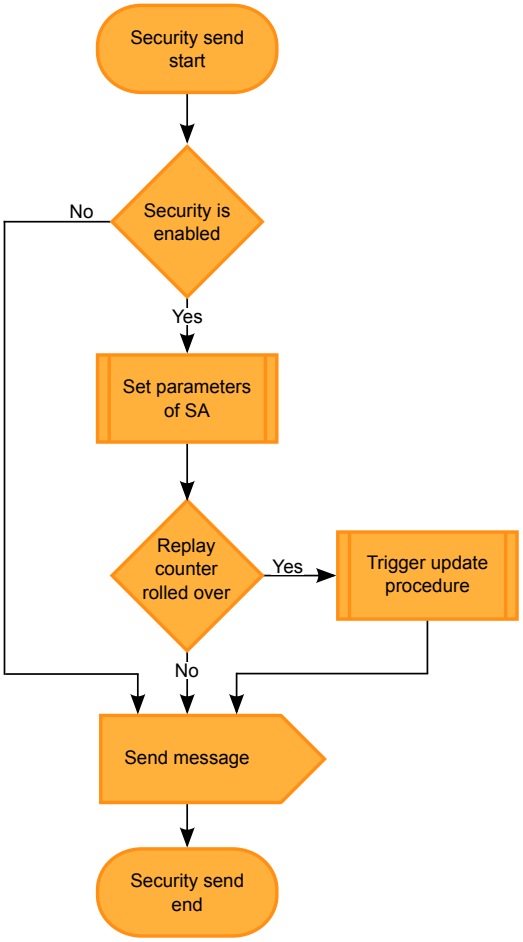

Figure 4.6: SDL diagram of the transmit process of the IEEE 1588v2 security extension

Initialization File

The initialization of the PTP stack is split into two parts – one general initialization for the PTP stack and a security related part. The general configuration is done via command line switches and it is already available by the PTP stack. A modification of this part introduces an additional switch for turning on and off the security extension.

The security related part consists of an initialization file. The configuration via command line is discarded due to the fact that the security module has to be fed with keys and timing data. This is an error-prone task and hardly to be achievable on command line due to the amount of data needed by the security module. Additionaly, the key data is equal for all nodes in one group, therefore a file offers the convenient possibility to distribute the configuration to several nodes easily.

The configuration file uses a Comma Separated Value approach, which offers a flexible structure for the configuration of the stack. The configuration offers five different labels, used to configure the security extension on startup. These values are given below together with an example entry in the configuration file and a short description.

enableSecurity: The *enableSecurity* label is used to enable different modes of the security extension. More information on TLV handling can be found in subsection 4.2.3. An entry in

Function Name	Return Value	Parameters	Description
addsaIncoming	void	saEntry *newEntry	Adds a new entry into the incoming Security Association Table, which passed as a parameter.
searchSaIncoming	saEntry *	PTP_PortId *portId	Searches for an entry with a specific port ID in the incoming Security Association Table, which is passed as a parameter.
setSaIncoming	void	–	Sets a pointer to the beginning of the incoming Security Association Table.
getSaIncoming	saEntry *	–	Returns a pointer to the current valid element in the incoming security association table.
nextSaIncoming	void	–	Sets the pointer of the incoming Security Association Table to the next element.

Table 4.1: Functions related to the handling and maintaining the incoming Security Association Table.

the configuration file would look like this: *enableSecurity,1*

tlv: The *tlv* label is used to add a TLV to a message with a specific ID and length. This is primarily for validation. More information on this can be found in subsection 4.2.3. An example for the configuration file would look like this: *tlv,abcd,10*

replaycounter: The *replaycounter* label is used to initialize the replaycounter. It can be set to a specific value. An example for the configuration file would look like this: *replaycounter,10*

timebetweenmessages: The *timebetweenmessages* label is used to configure the time, which has to elapse between the sending of two consecutive messages. The value is specified in µs. More information on this can be found in subsection 4.2.3. An example for the configuration file would look like this: *timebetweenmessages,25000*

key: The *key* label is used to add keys to the key table list. The configuration for this label has the following properties:

- An unique identifier for the key.
- The algorithm ID, which has to be used together with this key.
- The key itself in hexadecimal notation.
- And a time interval in seconds when the key expires.

An example for the configuration file would look like this: *key,1,2,password,1200;*

4.2.3 Verification and Validation

After the implementation, the system has to be verified and validated. Without this additional steps, it cannot be ensured that the stack works as it should and that it has inherent security breaches [TH09].

The first step is to use automatic tools to analyze the produced code. The tool used for this task is Valgrind (http://valgrind.org/). Valgrind is released under the terms of the of the GNU General Public License. It is a set of tools, which automatically detects bugs in the memory management and has a profiling functionality.

During the examination of the code several cases where pointed out where a reserved memory was not freed after the initialization. No bugs where found in the underlying structure. All bugs reported by Valgrind were corrected and, at the end, the system passed the checks without errors. Yet, such a static analysis cannot check the internal state machine and interact with dynamic processes during real operation.

For the verification of the internal logic, several tests have been developed. These tests can be launched from a modified stack. The behavior is changed with switches in the configuration file of the security extension (see subsection 4.2.2).

Thirteen test cases are defined to examine the development and correct behavior. Additionaly, these test cases can also be used for interoperability testing, as test pattern generator against other PTP stacks.

Table 4.2: Possible test cases of the implemented PTP stack

Test Case	Description
0	Disables the security extension. With this option it can be tested if the addition of the security extension has influence on the rest of the stack.
1	Enables the security extension and represents the normal mode of operation. This option is only available for the implemented stack and cannot be tested with other implementations.
2	Disables the security flag in the header of the PTP message. An implementation running in a secure operation mode has to discard this message immediately.
3	Does not append the mandatory Authentication TLV to the PTP message. This message qualifies as malformed, an implementation running in a secure operation mode has to discard this message immediately.
4	Inserts a wrong Integrity Check Value (ICV) in the Authentication TLV of the PTP message. The receiver calculates the ICV over the PTP message; the comparison of the received ICV and the calculated result differs and an implementation running in a secure operation mode has to discard this message immediately.
5	Keeps the replay counter at a fixed number. The standard specifies that a received ICV has to be greater than the one saved in the Security Association Table, therefore a check reveals that the replay counter is equal or smaller than the one saved. An implementation running in a secure operation mode has to discard this message immediately.

Test Case	Description
6	Sets the replay counter to a very high value, short before an overflow occurs. The replay counter is a 32 bit variable, therefore it overflows at 4294967295. Setting the replay counter to a value before this number forces the stack to trigger an update procedure to distribute the new next key ID and next lifetime ID. The replay counter is set via the configuration file, see subsection 4.2.2.
7	Sets the key ID in the Authentication TLV to an invalid number. An implementation running in a secure operation mode has to discard this message immediately.
8	Sets the lifetime ID in the Authentication TLV to an invalid number, neither the current lifetime ID nor the next lifetime ID. An implementation running in a secure operation mode has to discard this message immediately.
9	Sends a wrong response nonce in the response-request message to the receiver. The receiver compares the received value to the one saved in the Security Association Table. Due to the fact that the sender on purpose sent the wrong nonce and the check does not work out, the receiver has to discard the message immediately and the Security Association is never changed to the trusted state.
10	Sends a wrong response nonce in the response message to the receiver. The receiver compares the received value to the one saved in the Security Association Table. Due to the fact that the sender sent the wrong nonce on purpose and the check does not work out, the receiver has to discard the message immediately and the Security Association is never changed to the trusted state.
11	Floods the network with messages. When the stack runs as master, it sends constantly sync messages to the slaves. When the stack runs as slave, it sends constantly delay messages to the master. The interval between two consecutive messages is defined in the configuration file of the security extension (see subsection 4.2.2).
12	Adds an extra TLV to the existing PTP messages. The TLV is inserted before the Authentication TLV, therefore it is included in the security checks and should not produce an security related error. In this test, two or even three TLVs are incorporated in one message. These messages are passed on to the PTP stack, which has to handle the TLVs according to their identifier. If it cannot handle them it has to discard the TLV.
13	It is similar to twelve, it also appends an extra TLV. However, this time no Authentication TLV is appended. During the stack testing it might be possible that specific TLVs trigger a wrong behavior, therefore it is necessary to be able to send modified PTP messages without the appended security.

4.3 IPsec Protection

Secure communication on the IP level has many faces, but one of the most widespread protocols is IPsec. A lot of applications in different domains make use of this protocol, e.g., building Virtual Private Networks (VPN), connecting two servers via a secured connection, or offering secured connections for employees from home or a remote working place. In an environment where the traffic is already protected by IPsec, it is not useful to secure the PTP traffic with the security extension provided by PTP itself. The PTP traffic can use the already provided secured communication channel and reduce the necessary maintenance effort. As an alternative to the implemented secure PTP stack, the IPsec protocol stack is examined to show the influence on PTP.

4.3.1 Structure

IPsec is part of the network subsystem and does not have to be implemented. The IPsec protocol is fully integrated in the IP stack of the kernel. IPsec is used to compare the native solution of IEEE 1588v2 with a well established communication protocol. The information flow in the kernel can be in general divided into a receiving path and a transmitting path. Figure 4.7 depicts the logic flow of incoming and outgoing IP packets, together with the IPsec extension. The two communication paths, receiving and transmitting, are connected with a forwarding path. These functions are used to forward packets to other recipients and not to the local host.

IPsec Receive

A received IP packet is enqueued in the IP stack of the kernel. The first handlers of the packet are the ip_rcv and ip_rcv_finish functions, depicted in the left part of Figure 4.7. Standard unencrypted traffic, which has to be forwarded, can be directly pushed into the forwarding path. Traffic, which is encrypted or destined for the local host, is handled by the ip_local_deliver function. If an IPsec structure is detected, it is fed to the xfrm function, xfrm4 for IPv4 and xfrm6 for IPv6. This function handles either IPsec Authentication Header (AH) packets via ah_input, or IPsec Encapsulationg Security Payload (ESP) packets, with esp_input. The security features are checked and proper measurements for authenticating and decrypting the packet are taken. After this procedure, the packet is passed back to local delivery. If the packet is intended for local delivery, it is passed down to ip_local_deliver, which serves it to the next higher layer. The second possible way for the plain IP packet is the forwarding path. For this procedure, the packet is enqueued again at the beginning of the IP processing.

IPsec Transmit

Transmitting an UDP IP packet starts with enqueueing the packet in ip_push_pending_frames, depicted in the right part of Figure 4.7. If a packet is secured by IPsec, it is forwarded to the appropriate functions, ah_output or esp_output. In ah_output the data is authenticated by the authentication algorithms. The function esp_output performs the encryption of the data. The result, authenticated or encrypted packet, is fed back into ip_push_pending_frames. For a combined mode ESP and AH the packet is again forwarded to the specific IPsec function to perform the task. After the IPsec procedures, the packet is forwarded by ip_push_pending_frames to

ip_output, which is also the entry point for forwarded packets. The ip_output function relays the packet to ip_finish_output, which forwards it to the next lower layer.

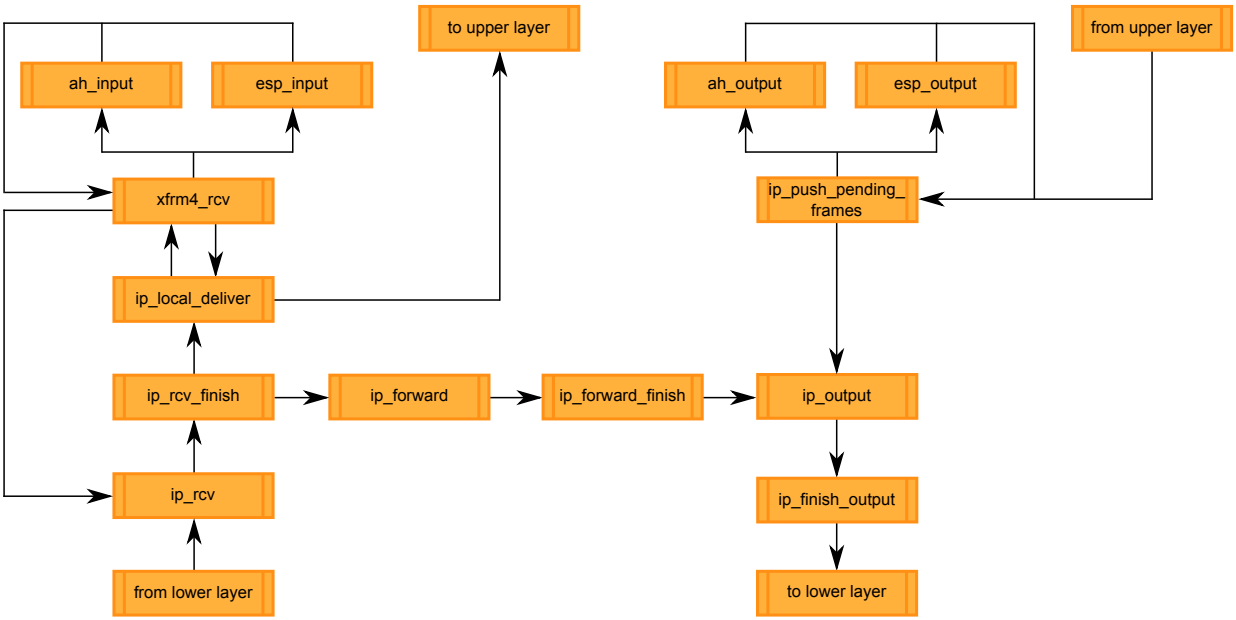

Figure 4.7: IPsec data flow in the Linux kernel

4.3.2 Interfaces

IPsec itself is transparent for applications, therefore the implementation of the application does not need to be modified compared to stock IP. Applications send the traffic as with non-secured IP and do not have to know anything about the underlying security enhancement provided by IPsec. The kernel uses two different structures for handling IPsec connection, Security Policies (SPs) and Security Associations (SAs).

Security Policies identify which traffic is handled by the AH functions, ESP functions, or both. These Security Policies are managed in the Security Policy Database (SPD). The Security Policy contains the IP address of the node, communicating with the local host, the modes and services used for the connection.

Besides the SP, which specifies the properties of the connection, two Security Associations are created, which specify how the traffic is handled. One SA is used for the incoming traffic and one SA for the outgoing traffic. The IP stack of the Linux kernel needs information for decrypting, encrypting, and authenticating the packets. This security related data is retrieved from the Security Association (SA) associated with the connection. For one host, several SAs can be provided and all of these associations are managed in the Security Association Database (SAD). An SA is composed of a source IP address, a destination IP address, the protocol type (AH, ESP or a combination of both), algorithms, keys, an unique identifier, and the Security Parameter Index (SPI). The SAs can be created in two different ways, either manual keyed connections or automatic keyed connections.

- Manual keyed connections: All information needed to setup the connection have to be provided manually, e.g., by an administrator. During the setup, it is specified which protocols,

algorithms and keys have to be used during runtime. The administrator has to setup the security associations and fill the SAD.

- Automatic keyed connections: A daemon has to setup the keys for the IPsec connection automatically. The Internet Key Exchange (IKE) protocol was designed to do this automatic update.

For setting up the connection, two programs are used in Linux, setkey and racoon. Both tools are part of the ipsec-tools software suite.

Setkey is a tool to administrate both, the Security Association Database and the Security Policy Database of the Linux kernel. It is possible to add, dump, flush, and update entries in the SAD. The configuration is either read from the standard input or read from a file.

Racoon is a tool, which makes use of the IKE protocol. The key management protocol is used to create Security Associations between two hosts. Racoon is called from the Security Policy Database.

5 Measurement Setup

After the security analysis of the clock synchronization stack in subsection 3, and the implementation which is described in the the previous section 4, the whole system is tested in a real network. The setup of the test network consists of two parts, first the hardware setup and second the software setup. Both parts are explained in detail in this chapter.

5.1 Setup of the Hardware

The components, which are used in the hardware setup, are consisting of three main parts:

1. Two industrial PCs.
2. Two hardware clock synchronization cards.
3. An oscilloscope or frequency counter.

The two industrial computers, depicted in Figure 5.1 in the rack, are low power devices with an Intel Celeron M CPU running at 800 MHz, 512 MB RAM. One industrial PC is equipped with a hard disk and the other one uses a flash drive for the operating system and data. The operating system installed on the nodes is a Linux operating system, Debian Lenny, with a 2.6.26 version of the kernel. One node is configured as PTP master and the other node is the PTP slave.

The hardware clock synchronization card is a PCI card (syn1588) with a crystal oscillator (50 ppm), which is installed in the industrial computers. The card is produced and sold by Oregano Systems. With this special hardware, a synchronization between two nodes in the two digit nano second range is possible. Furthermore, the card has four analog outputs, which can be configured depending on the application. For this setup, one of the ports is configured as a Pulse Per Second (PPS) port. The hardware clock of the card is generating one pulse at the beginning of each second. With such an output, it is possible to measure the synchronization between two independent nodes. The outputs of the two nodes are connected to an oscilloscope or a frequency counter (depicted in the middle of Figure 5.1), which are used to perform the measurements.

For verifying the test setup, an oscilloscope is used, which is a LeCroy Wave Pro 7300A with 30 Gigasamples (can be seen on the left side of Figure 5.1). The PPS outputs of the two Devices Under Test (DUT) are connected each to a channel on the oscilloscope to measure the time difference between the two signals. For the measurement sets, a highly accurate universal frequency counter

is used. The device is programmable and can be controlled externally. The settings are sent to the device, which then operates according to the commands. This ensures a consistent measurement setup, which delivers reproduceable results.

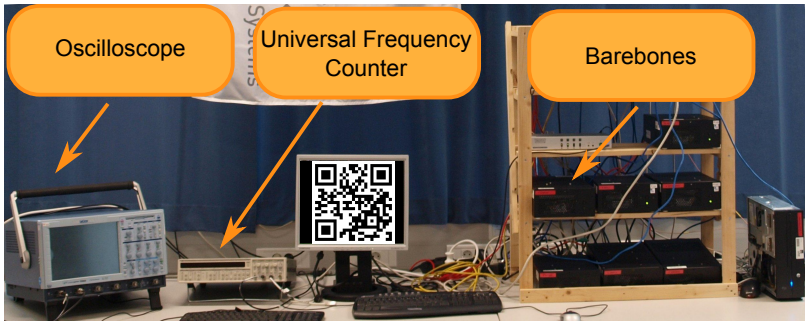

Figure 5.1: Measurement setup in the laboratory

5.2 Packet Delay Variation Measurement

The accuracy and variance of software clocks highly depends on the jitter introduced by the different parts present in the system. The network stack already present in the system also contributes to the jitter. In the Packet Delay Variation (PDV) measurements, the jitter of the software network stack in different scenarios is determined.

The first attempt for measuring the timestamps in the Linux kernel was based on kprobes[1]. This software is used to retrieve information from the kernel at specific positions. It is mostly used for debugging the kernel. As it turned out, the functionality of kprobes causes delays and adds additional jitter almost three magnitudes above the jitter of the measured delay. Therefore, the functions within the IP/IPsec stack were modified (instrumented) directly.

Figure 5.2 depicts the IP stack of the Linux kernel. The functions, which were instrumented, are marked with the clock box symbols. To determine the introduced delay, a timestamp is drawn at the beginning of the processing of an IP packet and a second timestamp is drawn when the packet leaves the IP stack.

In the receive path, the timestamp $t_{Receive1}$ is taken in the *ip_rcv* function, which is the point of entry from the lower layers. The second timestamp $t_{Receive2}$ is drawn when the packet exits *ip_local_deliver,* which passes the packet on to the upper layers.

In the transmit path, the timestamp $t_{Transmit1}$ is drawn when the packet from the upper layer enters the IP stack, which in case of UDP is in ip_push_pending_frames. The second timestamp $t_{Transmit2}$ is drawn at the end of *ip_finish_output.*

To keep the influence on the measurements as low as possible, the timestamps are recorded in a memory array. The jitter prone printing of the memory array to the kernel log file is done after the measurements are finished.

The instrumentation has to handle the fact that IPsec packets traverse the instrumented IP functions multiple times. To exemplify such a case, the receiving of an ESP packet in tunnel

[1] http://sourceware.org/systemtap/kprobes/

mode is explained. The packet is passed from the lower layers to *ip_rcv*. After that the packet passes *ip_local_deliver*. Now the packet is not handed over to the upper layer, but processed by *xfrm4_rcv*, which handles the IPsec part of the packet. After the outer IPsec header is processed and stripped, the packet is again enqueued in *ip_rcv*. The packet still has an IP header and is processed again. This cycle repeats as long as there are IP headers present and for every round two timestamps are generated. When all headers are processed, the packet is forwarded to the upper layers.

Figure 5.3 shows the measurement setup of a single node for PDV measurement. The application transmits/receives the packets and the Linux kernel processes them. The packets are then transferred over the network to the receiver/transmitter. This setup is used identically, one time with IP packets and the second time with IPsec packets. The data, which is stored in the kernel log, includes the identifier of the timestamp and the timestamp itself.

Because the logging is built into the kernel, it starts immediately after the system start and contains a lot of irrelevant measurement data. The actual measurement starts with a sequence, which can be identified in the log file, therefore it is possible to start in determined state. The analysis of the data depends on the mode:

1. IP, one round per packet, two timestamps
2. AH or ESP in transport mode, two rounds per packet, four timestamps
3. AH or ESP in tunnel mode, three rounds per packet, six timestamps
4. ESP+AH in transport mode, three rounds per packet, six timestamps
5. ESP+AH in tunnel mode, four rounds per packet, eight timestamps

The first measurement of plain IP without security serves as a reference for all measurements. The Precision Time Protocol (PTP) operates on the IP layer and, therefore, no dedicated PDV measurement for this protocol is needed.

5.3 Measurement of Clock Synchronization Precision

The PDV measurements put a focus on the delay, which is introduced by the different security measures. This delay is introduced in lower layers and the control algorithm, introduced by the PTP application, tries to minimize such effects. The aim of this measurements is to compare the results between the different implementations. Furthermore, it is possible to compare the PDV measurement, described in the previous subsection 5.2, to the clock synchronization quality of the synchronized stack.

The setup can be divided into three classes.

1. PTP over IP
2. PTP over IP together with IPsec
3. PTP over IP with native IEEE 1588v2 security extension

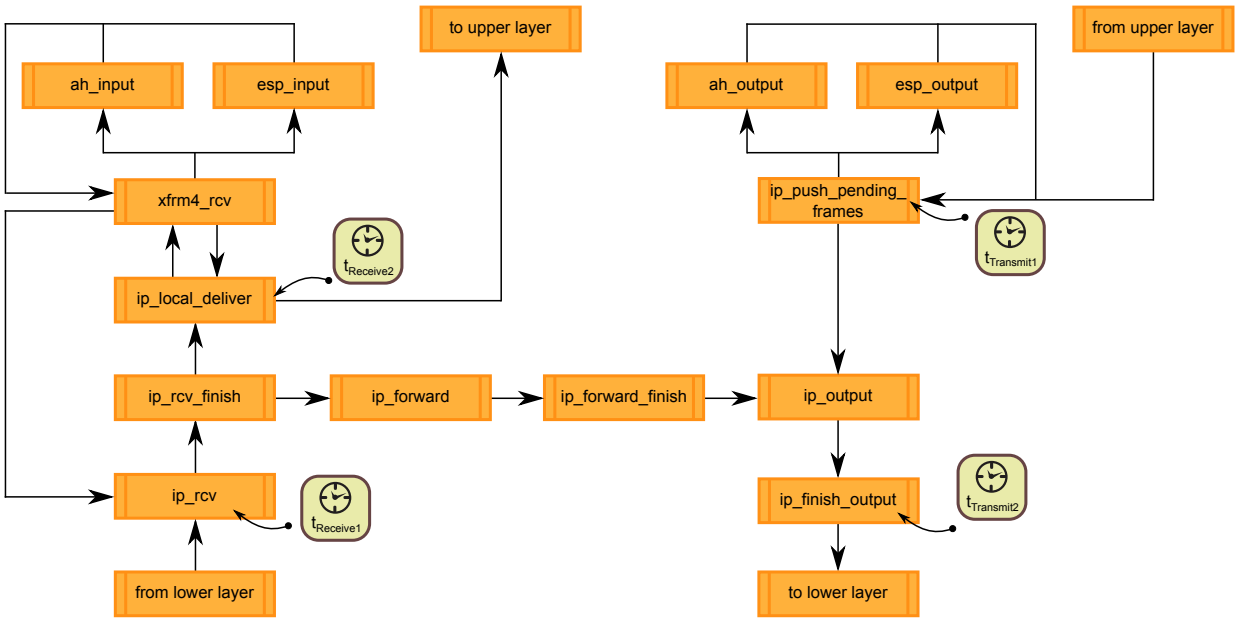

Figure 5.2: Instrumentation points in the IP/IPsec stack of the Linux kernel

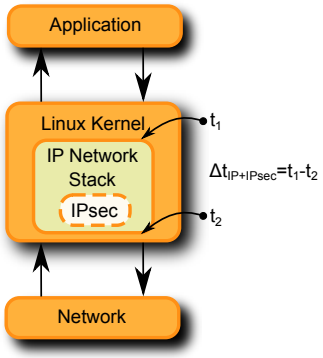

Figure 5.3: Position of timestamps in the measurement setup

5.3.1 IEEE 1588v2 Setup

The configuration of the security module for the PTP stack is provided via a configuration file, the configuration is static and for all nodes the same. The configuration used in the setup:

```
enableSecurity,1
#tlv,0xidentifier(in hex),length (complete tlv)
#must be activated via switch in the line above
tlv,0xabcd,10
#identifier,replay counter
#counteroverflow at 4294967295
replaycounter,10
#time between messages in us
#timeBetweenMessagesint,value timebetweenmessages,25000
#identifier,keyId,algorithmId,key,timeinterval
```

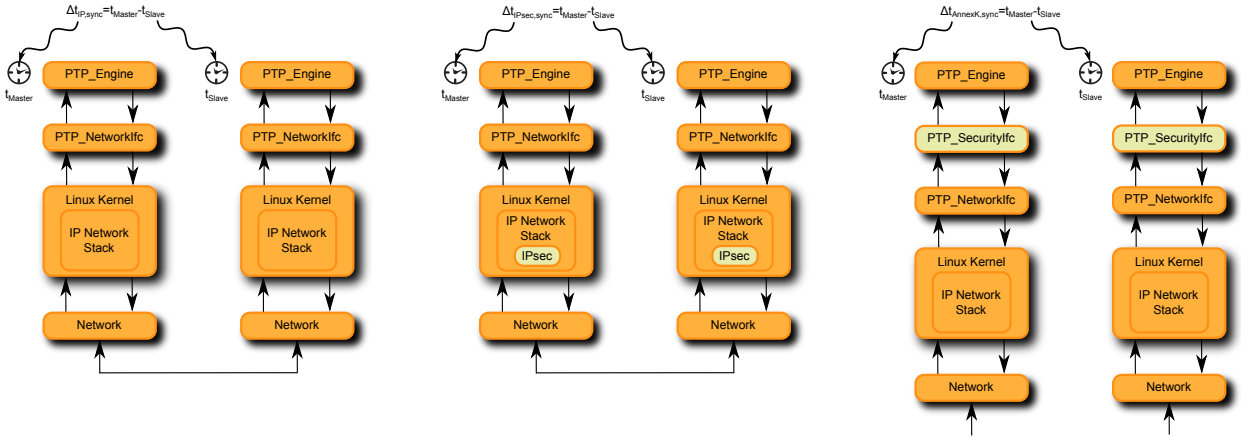

Figure 5.4: Delay measurements in three different setups: IP, IPsec, and IEEE 1588v2 Annex K

```
#keyId must be unique
#algorithmId 1 for sha1 or 2 for sha256
#key in hex
#timeinterval key expiration time in s
key,1,1,abcdabcdabcdabcd,20000
key,2,2,1234567890123456,20
key,3,1,1234567890abcdef,20
key,4,2,1234abcd1234abcd,20
```

The configuration of the PTP stack is also static, but it is provided at runtime. The stack is forced to use a specific network interface and the synchronization interval is set to one message per second.

5.3.2 IPsec Setup

To be able to use all services and modes of IPsec, the appropriate infrastructure has to be setup. Three tools have to be prepared for the measurement setup:

1. Certificate Authority (CA)
2. setkey
3. racoon

Each of the tools has to be configured and separately brought up.

Key Distribution

For the IPsec implementation in Linux, certain modes can be used together with key distribution schemes. These schemes offer automatic key negotiation between the communication partners. This key distribution needs a Certificate Authority (CA), which distributes digital certificates. With a digital certificate, the authenticity of a public key can be identified without sending

private information from the user over the underlying network. The private key remains secret to the owner and is not published. Third parties rely on this certificate, which is linked to a specific public key. This security concept depends on trusted relationships. The CA acts as a trusted third party and all communication partners trust this third party. A communication partner, which is not certified, cannot participate in the communication. The abilities of the CA is not only to issue digital certificates, but it can also revoke certificates. This action invalidates an already trusted certificate. The structure of an CA, which is used in this setup, is depicted in Figure 5.5. The Certificate Authority signs two certificates for the involved communication partners. Because both entities trust the CA, the entities trust each other and can exchange information.

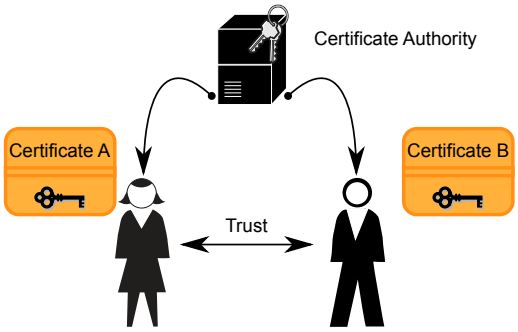

Figure 5.5: Structure of a Certificate Authority

In the test setup, the task of the CA are simple: Each of the clients requires a set of credentials, which are used to authenticate themselves and the other communication partners. The setup of the CA starts with the generation of a root key and a self signed certificate. Each node in the network needs a certificate which is certified by the root CA. After every certificate is signed, the nodes have all the necessary keys and certificates to establish a connection.

Setting IPsec rules

The functions provided by setkey are setting and deleting rules, which have to be applied to the IP packet, e.g., AH, ESP, or any combination of these. It also defines on which connections these rules have to be applied to. This properties can be applied to traffic of a specific node or a complete network. It is also possible to define different rules for sending and receiving information. An example for such a configuration file in the test network is given in the following:

```
flush;
spdflush;
spdadd 192.168.105.15 192.168.105.25 any -P in  \
 ipsec esp/transport/192.168.105.15-192.168.105.25/require;
spdadd 192.168.105.25 192.168.105.15 any -P out \
 ipsec esp/transport/192.168.105.25-192.168.105.15/require;
```

The first two lines of the configuration file are deleting all rules from the existing configuration. This is done to start from a clean state. After that, a rule for the incoming traffic is generated. Traffic, which is destined from the local interface (192.168.105.15) to the remote interface (192.168.105.25), has to use IPsec in ESP-transport mode. The next line takes care of the outgoing

traffic similar to the configuration of the incoming connection. For the test setup, this configuration file is provided at the start up of setkey and takes care of all necessary configurations needed.

Key Exchange for IPsec

Racoon is a daemon for automatically keying IPsec connection. This service is provided with the help of the Internet Key Exchange (IKE) protocol. In the configuration file of racoon (racoon.conf) it is specified which certificates have to be used and which algorithms are supported for the data exchange. An example for such a configuration file for the test setup looks as follows:

```
path include "/etc/racoon";
path_certificates "/root/ipsec/certs";
path pre_shared_key "/root/ipsec/psk.txt";
path script "/etc/racoon/script";
listen { isakmp 192.168.105.15 [500]; }
remote 192.168.105.25 { ... }
sainfo anonymous { ... }
```

This daemon can be used in all modes and for all services provided by IPsec. For the test setup the key exchange is always the same for all IPsec measurements.

6 Measurements and Results

The test setup described in chapter 5 is used for two different measurements, Packet Delay Variation (PDV) and an overall clock offset measurement.

Before the actual measurement data is discussed, different influences on the measurement setup are investigated and analyzed. The first part of the measurements is putting the focus on the Packet Delay Variation, introduced in the network stack of the Linux kernel. Therefore, the Linux kernel was modified, as outlined in the previous section, 5.2. The measurement setup captures 10000 values to have a profound statistical base. These values are used to calculate a mean value and the standard deviation for the packet delay introduced by the Linux kernel. The baseline of the measurements is represented by plain IPv4, all other measurements of this series are compared to this first measurement cycle. The complete series is composed of the following measurements:

- IPv4
- IPv4 with Authentication Header in transport mode
- IPv4 with Authentication Header in tunnel mode
- IPv4 with Encapsulating Security Payload in transport mode
- IPv4 with Encapsulating Security Payload in tunnel mode
- IPv4 with Encapsulating Security Payload combined with Authentication Header in transport mode
- IPv4 with Encapsulating Security Payload combined with Authentication Header in tunnel mode

The second part of the measurements puts the focus on the synchronization between two clock synchronization nodes. The PTP application runs with different network layer configurations:

- IPv4 with PTP, running without native security extension
- IPv4 and PTP, running with the native IEEE 1588v2 security extension
- IPv4 with Authentication Header in transport mode, running PTP without native security extension

- IPv4 with Authentication Header in tunnel mode, running PTP without native security extension
- IPv4 with Encapsulating Security Payload in transport mode, running PTP without native security extension
- IPv4 with Encapsulating Security Payload in tunnel mode, running PTP without native security extension
- IPv4 with Encapsulating Security Payload combined with Authentication Header in transport mode, running PTP without native security extension
- IPv4 with Encapsulating Security Payload combined with Authentication Header in tunnel mode, running PTP without native security extension

Plain IP without security and with PTP is used as a reference. For the test setup a master and a slave synchronize their clocks and the offset between the two clocks is recorded. These values are used to calculate a mean value and the standard deviation of the offset.

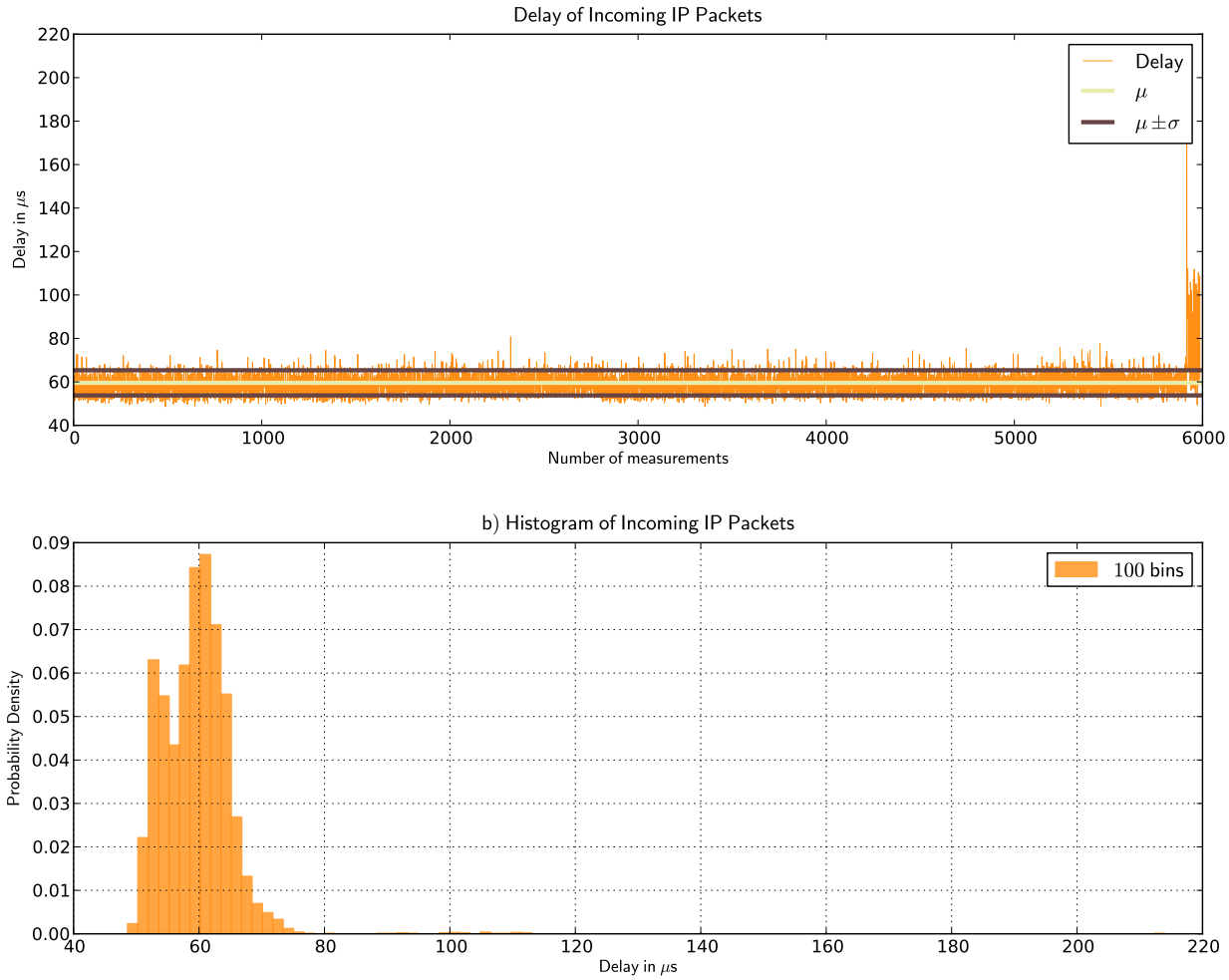

Figure 6.1: Influence of the system usage on the Packet Delay Variation

6.1 Influence on Measurement Results

First measurements showed that several factors have a severe influence on the Packet Delay Variation (PDV) and the clock synchronization. During the analysis of the measurement data, three main factors where found, which had the greatest impact.

The first influencing factor was the correlation between the system load and the PDV. Figure 6.1 shows the measurement cycle of the packet delay of a system. At the end of the measurement, an additional load on the system was applied. The amount of values at the end of the measurement cycle are too few to be seen in the histogram, but are present. Therefore both representations are necessary to see effects on the measurement. The load on the system consisted of typing some characters on an empty shell. The sudden change of the PDV is clearly visible at the end of the Figure 6.1. For the measurements which were taken, a complete autonomous setup was developed to prevent external influences on the measurements. Network services not needed for the measurements were deactivated and during the measurements no interaction with the system was performed.

The second factor which had great influence on the clock synchronization, was the synchronization cycle of the hard disk. Figure 6.2 shows the measurement cycle of the offset between a master and a slave. In the middle of the measurement cycle the system performed a harddisk synchronization. The system has to write to the disc and still keep up with the reqests, which are performed via the network interface. This event depends on the filesystem and the system settings, which are used on the system. The measurement setup uses the ext3 filesystem. The default value for the ext3 synchronization interval is five seconds. To minimize the effect on the synchronization the default value was raised to 15 minutes. During the measurements, the synchronization was still clearly visible with a high peak, shown in Figure 6.2. For further measurements, the harddisk synchronization interval was raised further, to ensure that this event does not interfere with the measurements.

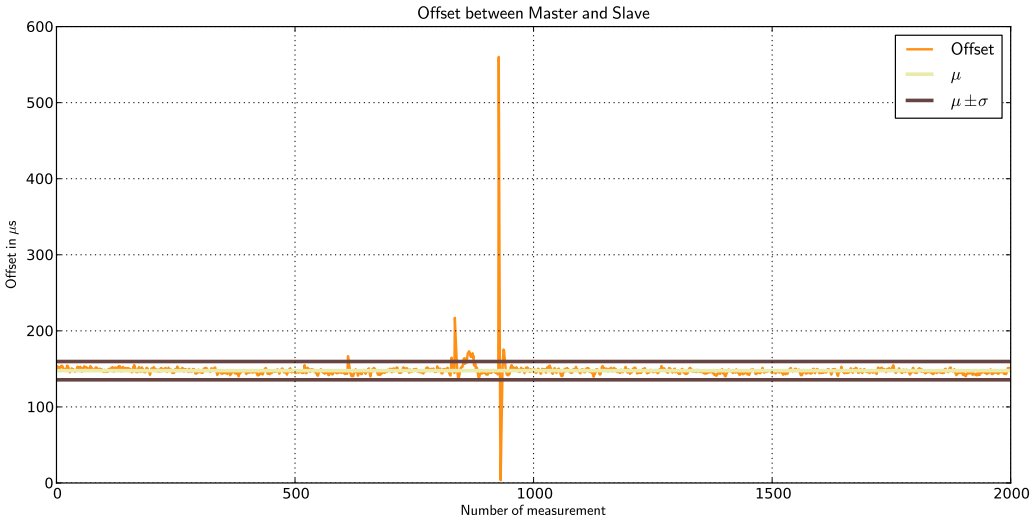

Figure 6.2: System synchronization of the harddisk (peak in the offset)

The third factor is the transient behavior on the startup. Figure 6.3 shows the measurement cycle of the offset between a master and a slave. At the beginning of the measurement, the transient behavior can be seen. Approximately 600 seconds after the stack is started, the control loop has

settled and it oscillates around a fixed value. For the measurement of the clock synchronization, the analysis of the measurement values was limited to the values after the control loop has settled, to get the data from an already running system and not from the start up.

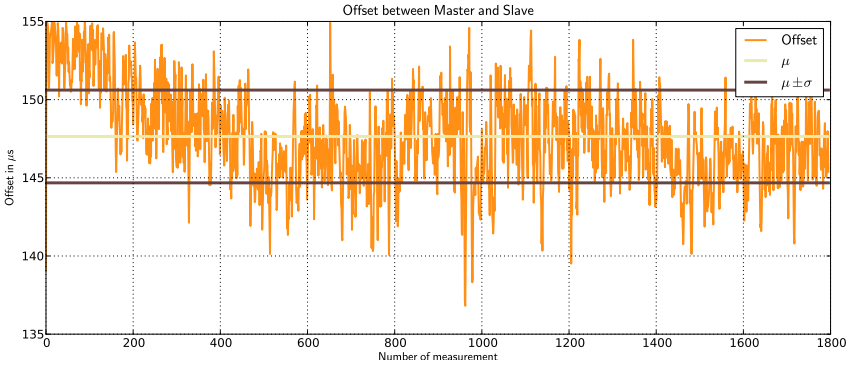

Figure 6.3: Startup of clock synchronization

6.2 Packet Delay Variation Measurement

All the diagrams shown in this section follow the same logic. The upper part of the diagrams shows the delay of the incoming packets received from the network. The lower part of the diagrams shows the delay of the outgoing packets sent to another node in the network. The left subfigures show the packet delay of the received or sent packets and the right subfigures depict a histogram of the packet delay.

The results show distinctive distributions, which accumulate around one or two points. Even after extensive resarch, no correlation between the measurements and the specific values of the operating system could be found. Possible candidates for the correlation are given in Table 6.1.

Type	Frequency	Time
CPU frequency	800 MHz	1.25 ns
Clock oscillator	1 kHz	1 ms
Scheduler 1	200 Hz	5 ms
Scheduler 2	20 Hz	50 ms
Scheduler 3	10 Hz	100 ms
Scheduler 4	1.67 Hz	600 ms
Scheduler 5	1.25 Hz	800 ms
Task priority 1	100 Hz	10 ms
Task priority 2	4.76 Hz	210 ms
Sync filesystem	8.3 mHz	2 min

Table 6.1: System events

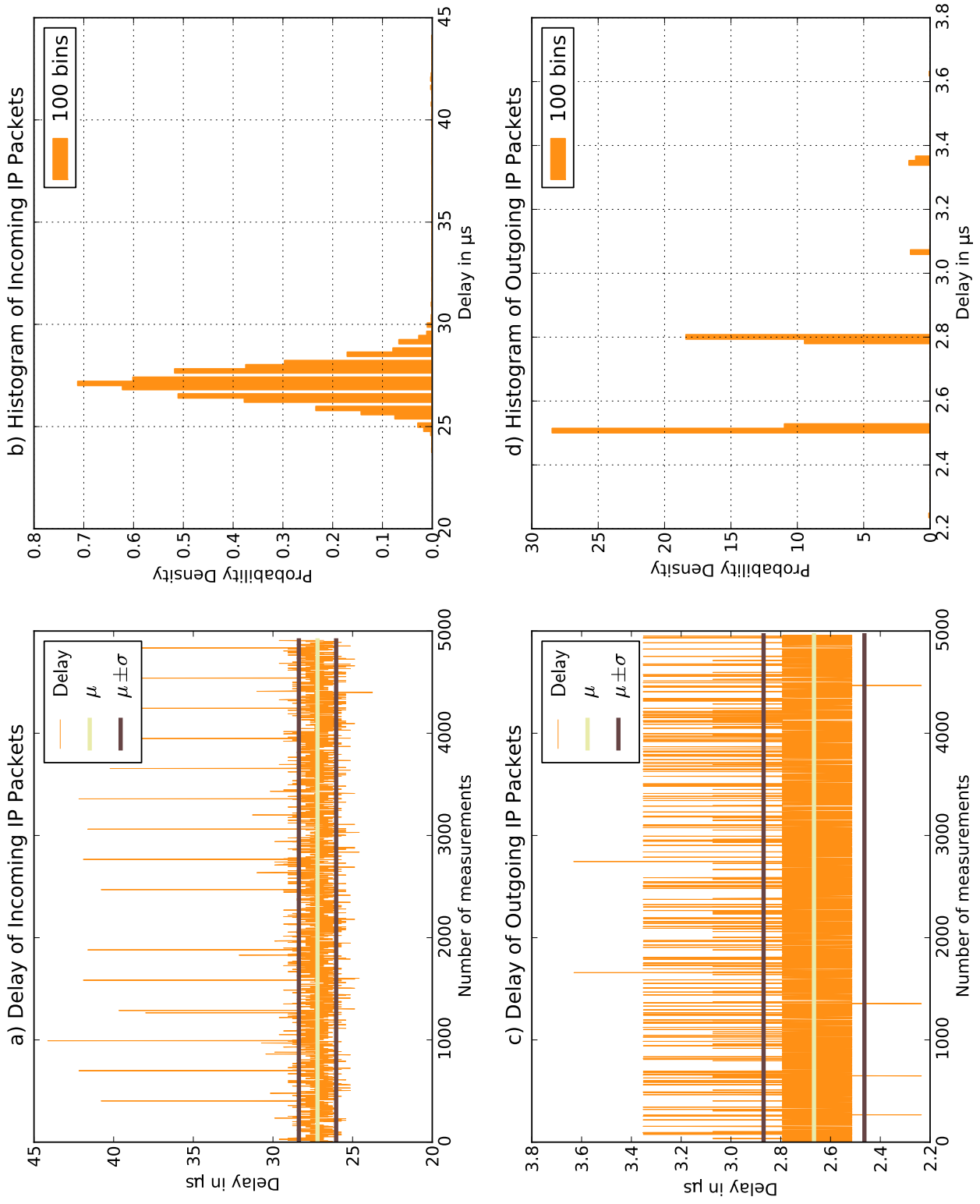

Figure 6.4: Packet delay variation of IPv4 packets without security

6.2.1 IPv4 without Security

The measurement of the delay of IPv4 packets is depicted in Figure 6.4a to 6.4d.

Figure 6.4a shows the delay over the measurement cycle for incoming packets. Most values are residing in a band around 27 µs, but several outliers can be clearly spotted. These values are 1.62 x the base values. The system used for the measurements is an off-the-shelf operating system, which does not offer real-time services. The scheduling has a great influence on the latency. These task switches introduced by the hardware latencies also add jitter to the packet delay.

The histogram of the incoming delay is shown in Figure 6.4b. One local maximum can be seen, at approximately 27 µs.

Figure 6.4c shows the delay over the measurement cycle for outgoing packets. Most values are residing in a band around 2.7 µs, but a lot of outliers are visible. In contrast to diagram (a), in diagram (c) the measurement values have a distinct cut off characteristic. This property is visible due to the changed scale on the y-axis.

The histogram of the outgoing delay can be found in Figure 6.4d. In contrast to the histogram of the incoming delay, in the histogram of the outgoing delay the quantization of the values is clearly visible. The values have a quantization of approximately 200 µs.

6.2.2 IPv4 Authentication Header in Transport Mode

The measurement of the delay of IPv4 packets with Authentication Header in transport mode is depicted in Figure 6.5a to 6.5d.

Figure 6.5a shows the delay over the measurement cycle for incoming packets. Most values are residing in a band around 80 µs, but again several outliers can be found. The ratio between the base values and the maximum outliers has increased in comparison to plain IP without security (Figure 6.4a).

The histogram of the incoming delay is shown in Figure 6.5b. There is one local maximum present at approximately 80 µs.

Figure 6.5c shows the delay over the measurement cycle for outgoing packets. Most values are residing in a band around 16 µs, the outliers are again clearly visible. In contrast to Figure 6.4c, the maximum of the outliers is over 14 times higher.

The histogram of the outgoing delay can be found in Figure 6.5d. Similar to IPv4 without security enhancements, the outgoing delay has one local maximum.

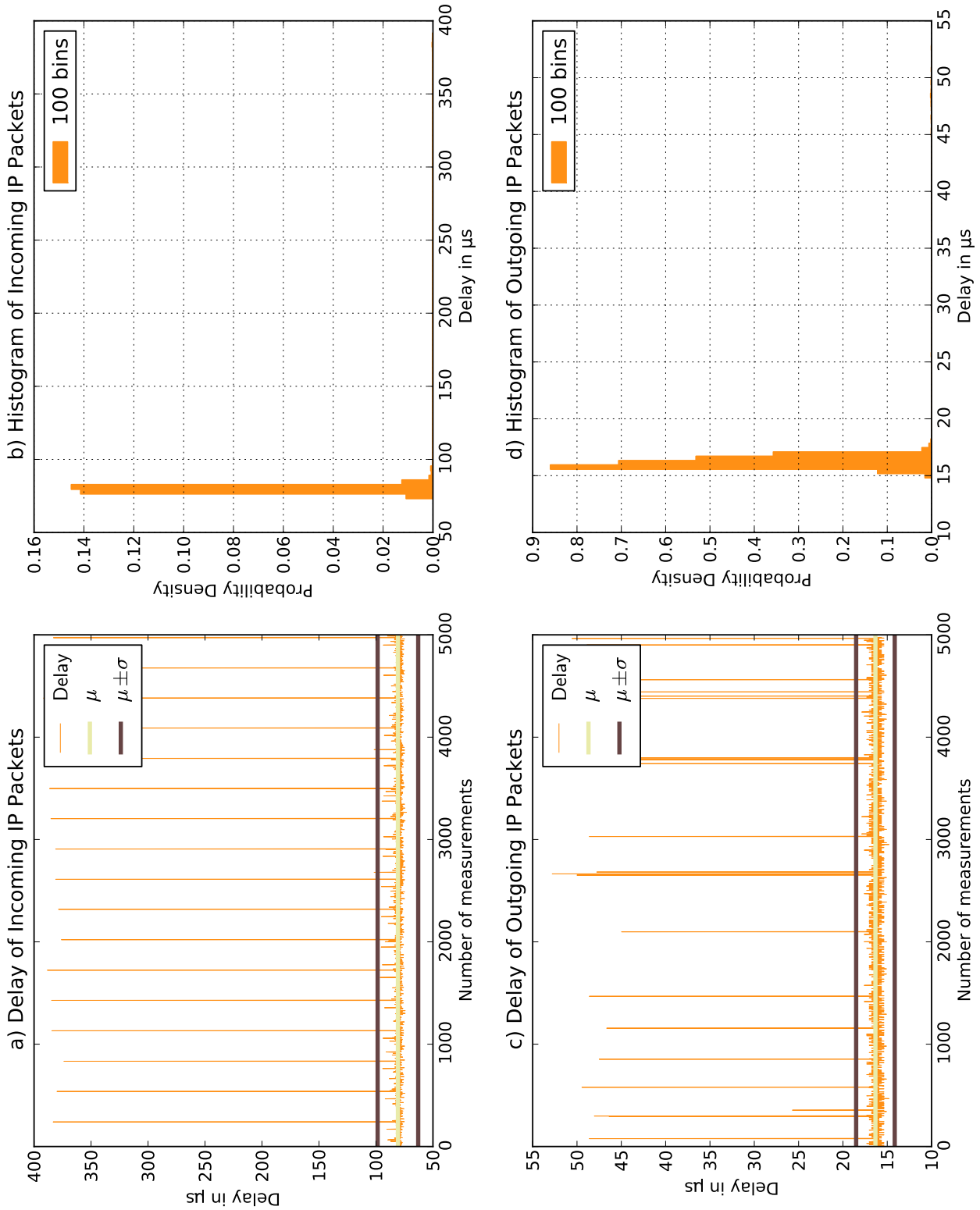

Figure 6.5: Packet delay variation of IPv4 packets with Authentication Header in transport mode

6.2.3 IPv4 Authentication Header in Tunnel Mode

The measurement of the delay of IPv4 packets with Authentication Header in tunnel mode is depicted in Figure 6.6a to 6.6d.

Figure 6.6a shows the delay over the measurement cycle for incoming packets. Most values are residing in a band around 84 µs. This value is a little bit higher than in the transport mode of the Authentication Header. Again, several outliers can be found. The ratio between the base values and the maximum of the outliers has increased in comparison to Figure 6.4a and is similar to Figure 6.5a and it is now approximately 4.76 x of the base values.

The histogram of the incoming delay is shown in Figure 6.6b. Similar to the previous two cases, one local maximum is present. It is located at approximately 84 µs.

Figure 6.6c shows the delay over the measurement cycle for outgoing packets. Most values are residing in a band around 18 µs. The maximum values of the outliers are higher than in the case of IPv4 with Authentication Header in transport mode and the ratio between the maximum of the outliers and base values is approximately 3.2.

The histogram of the outgoing delay can be found in Figure 6.6d. This time the histogram can also be compared to IPv4, which has one maximum. The difference between the two is that the peak for IPv4 with Authentication Header in tunnel mode is almost 15 times higher. The two modes, transport and tunnel, of the Authentication Header have a similar delay.

6.2.4 IPv4 with Encapsulation Security Payload in Transport Mode

The measurement of the delay of IPv4 packets with Authentication Header in tunnel mode is depicted in Figure 6.7a to 6.7d.

Figure 6.7a shows the delay over the measurement cycle for incoming packets. Most values are located around 104 µs. This value is higher compared to the tunnel mode of the Authentication Header and also higher than the Authentication Header in transport mode. The outliers observed are comparable to the previous measurements. The ratio between the base values and the maximum of the outliers has decreased to approximately 4.

The histogram of the incoming delay is shown in Figure 6.7b. Again one maximum is present, it is at approximately 104 µs.

Figure 6.7c shows the delay over the measurement cycle for outgoing packets. The values are located close to 27 µs. The values of the outliers have also increased compared to IPv4 with Authentication Header in transport or tunnel mode.

The histogram of the outgoing delay can be found in Figure 6.7d. Again the histogram is comparable to IPv4 or IPv4 with Authentication Header in tunnel mode.

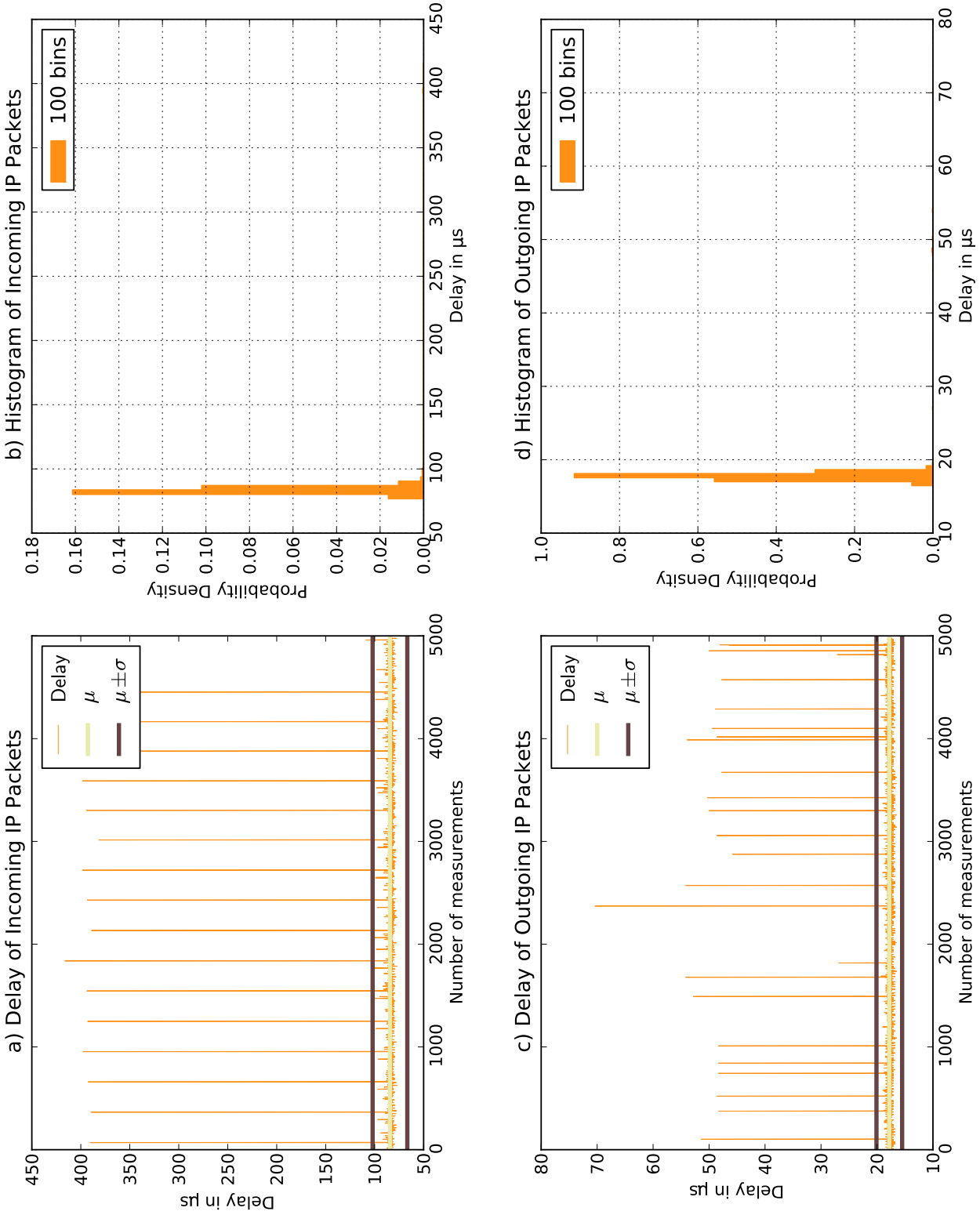

Figure 6.6: Packet delay variation of IPv4 packets with Authentication Header in tunnel mode

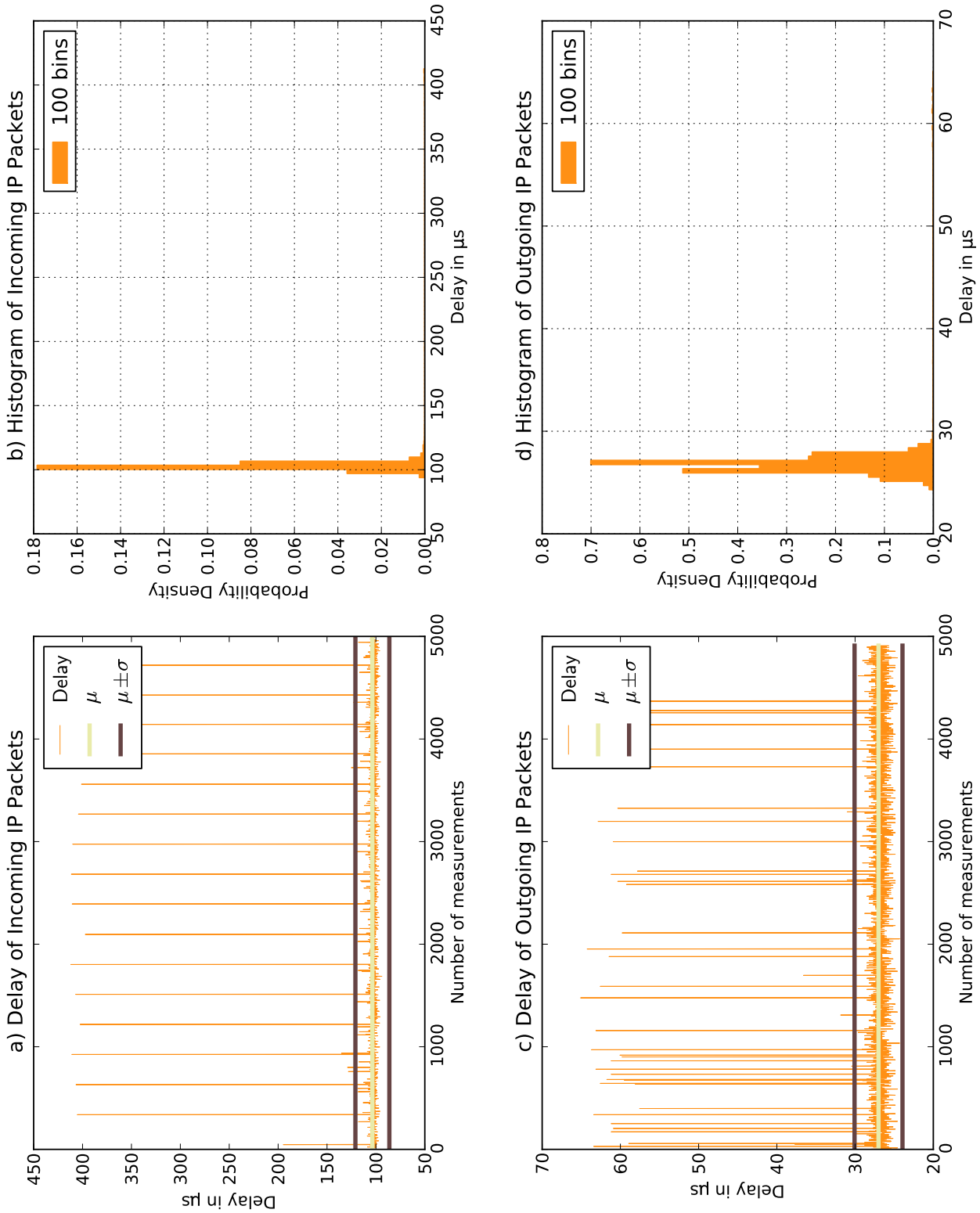

Figure 6.7: Packet delay variation of IPv4 with Encapsulation Security Payload in transport mode

6.2.5 IPv4 with Encapsulated Security Payload in Tunnel Mode

The measurement of the delay of IPv4 packets with Encapsulated Security Payload in tunnel mode is depicted in Figure 6.8a to 6.8d.

Figure 6.8a shows the delay over the measurement cycle for incoming packets. Most values are residing around 103 µs. This value is comparable to ESP in transport mode. The absolute values of the outliers are increasing and is even more than ESP in transport mode, 420 µs. The ratio between the base values and the outliers has increased a little bit in comparison to the previous measurements. It is now a little bit over 4.

The histogram of the incoming delay is shown in Figure 6.8b. The local maximum is at 103 µs, similar to the previous case.

Figure 6.8c shows the delay over the measurement cycle for outgoing packets. Most of the values center around 26 µs. This value has a little bit decreased compared to ESP in transport mode.

The histogram of the outgoing delay can be found in Figure 6.8d. Also in this measurement there is only one peak at around 26 µs. Compared to IPv4, the delay has increased over 10 times.

6.2.6 IPv4 Encapsulating Security Payload and Authentication Header in Transport Mode

The measurement of the delay of IPv4 packets with Encapsulation Security Payload and Authentication Header in transport mode is depicted in Figure 6.9 6.9a to 6.9d.

Figure 6.9a shows the delay over the measurement cycle for incoming packets. Most values are residing in a band around 124 µs. This value has again increased in comparison to the previous measurement cycles. The outliers observed in the previous measurements are like the ones found in this measurement.

The histogram of the incoming delay is shown in Figure 6.9b. One local maximum is present at approximately 124 µs. This value has increased when compared to the previous measurements.

Figure 6.9c shows the delay over the measurement cycle for outgoing packets. Most values are residing around 35 µs. This is 15 times the delay, which has been measured in IPv4. The values of the maximum outliers has also slightly increased compared to previous measurements. The ratio between the base values and the maximum of the outliers is approximately 2.

The histogram of the outgoing delay can be found in Figure 6.9d. Again, the histogram can be compared to IPv4, which has only one maximum located close to 35 µs. The increase of 15 times compared to IPv4 is clearly visible in this diagram.

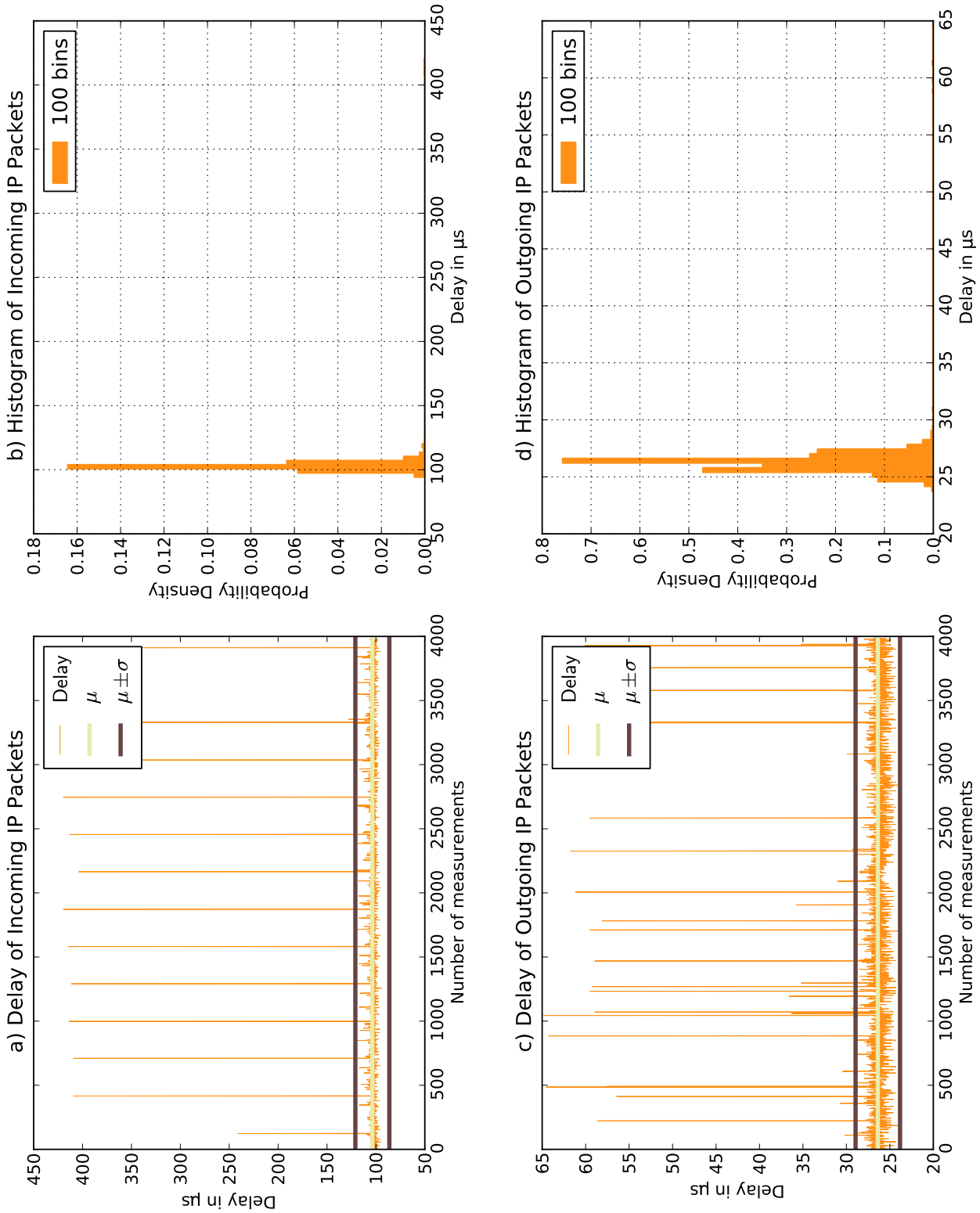

Figure 6.8: Packet delay variation of IPv4 with Encapsulation Security Payload in tunnel mode

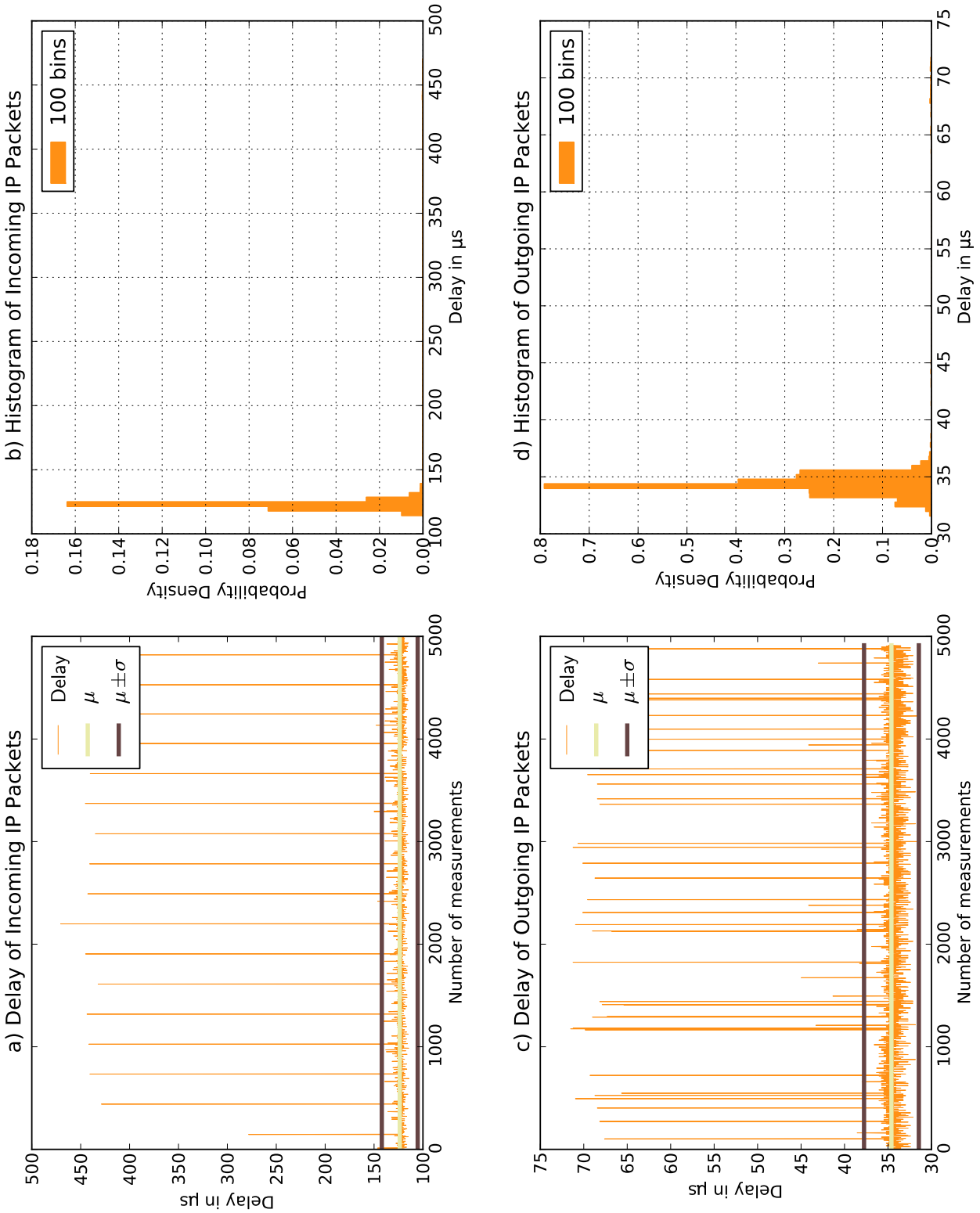

Figure 6.9: IPv4 Encapsulating Security Payload and Authentication Header in transport mode

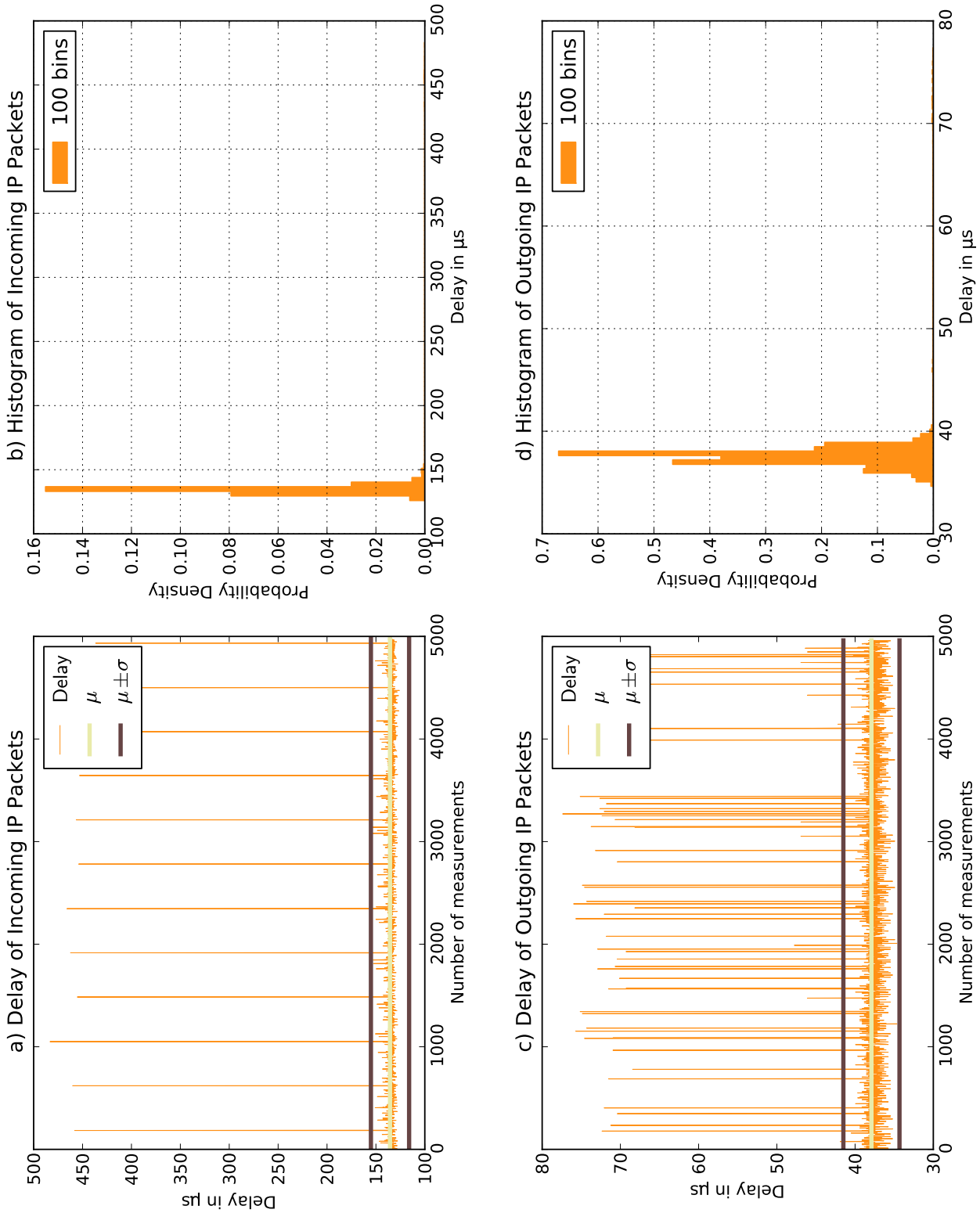

Figure 6.10: IPv4 Encapsulating Security Payload and Authentication Header in tunnel mode

6.2.7 IPv4 Encapsulating Security Payload and Authentication Header in Tunnel Mode

The measurement of the delay of IPv4 packets with Encapsulating Security Payload and Authentication Header in tunnel mode is depicted in Figure 6.10a to 6.10d.

Figure 6.10a shows the delay over the measurement cycle for incoming packets. Most values are residing around a mean value slightly below 136 µs. This is the highest value for all PDV measurement cycles. Both services, ESP and AH, are present and an additional header for tunnel mode is used, therefore this measurement cycle puts the highest stress on the CPU. Again, several outliers can be found, and these values are also the highest measured values over all PDV measurement cycles.

The histogram of the incoming delay is shown in Figure 6.10b. Also in this measurement, one local maximum is present, at approximately 136 µs.

Figure 6.10c shows the delay over the measurement cycle for outgoing packets. Most of the values are located a little bit under 38 µs. The amount of outliers is in the same range as in the previous measurements.

The histogram of the outgoing delay can be found in Figure 6.10d. This time the histogram is a little bit different. Similar to the previous measurement, e.g., Figure 6.9d, one local maximum can be found a little bit below 38 µs.

6.2.8 Summary of Packet Delay Variation Measurements

The aggregation of the measurement cycle for the PDV measurements can be found in Table 6.2. The results of the measurement cycle are divided into an input and an output section. Comparing the input and the output delay, a clear asymmetry is noticeable. The highest ratio between input and output delay can be found in IPv4. Here, the ratio is 10.19 and it is dropping with the increase of security services. It comes down to 3.57 for ESP and AH in tunnel mode and 3.58 for ESP and AH in transport mode.

IP configuration	Input ($\mu \pm \sigma$)	Output ($\mu \pm \sigma$)
IPv4	27.20 µs ±1.17 µs	2.67 µs ±0.20 µs
IPv4 AH Transport	80.73 µs ±17.79 µs	16.33 µs ±2.18 µs
IPv4 AH Tunnel	84.36 µs ±17.70 µs	17.80 µs ±2.29 µs
IPv4 ESP Transport	103.62 µs ±17.35 µs	27.00 µs ±3.06 µs
IPv4 ESP Tunnel	103.65 µs ±17.38 µs	26.39 µs ±2.56 µs
IPv4 ESP and AH Transport	123.72 µs ±18.34 µs	34.60 µs ±3.15 µs
IPv4 ESP and AH Tunnel	135.58 µs ±19.52 µs	37.91 µs ±3.59 µs

Table 6.2: Roundup of the PDV measurement results

Figure 6.11 shows the visualization of the PDV results. Starting with the reference measurement of plain IP without security for both cases, input and output, the least delay and standard deviation

are visible. The delay is represented by the distance of the center of the circle in the diagram and the standard deviation is represented by the area of the circle of the measurement. With every measurement, the absolute delay and the standard deviation are increasing constantly. The effect of the increase is more distinct at the incoming path and less distinct for the outgoing packets. Additionally, the measurement values for plain IP in the outgoing path are so small, they are invisible at this scale. Overall it can be observed that the addition of IPsec headers and the increased use of cryptographic functions increases the PDV for both pathes, incoming and outgoing.

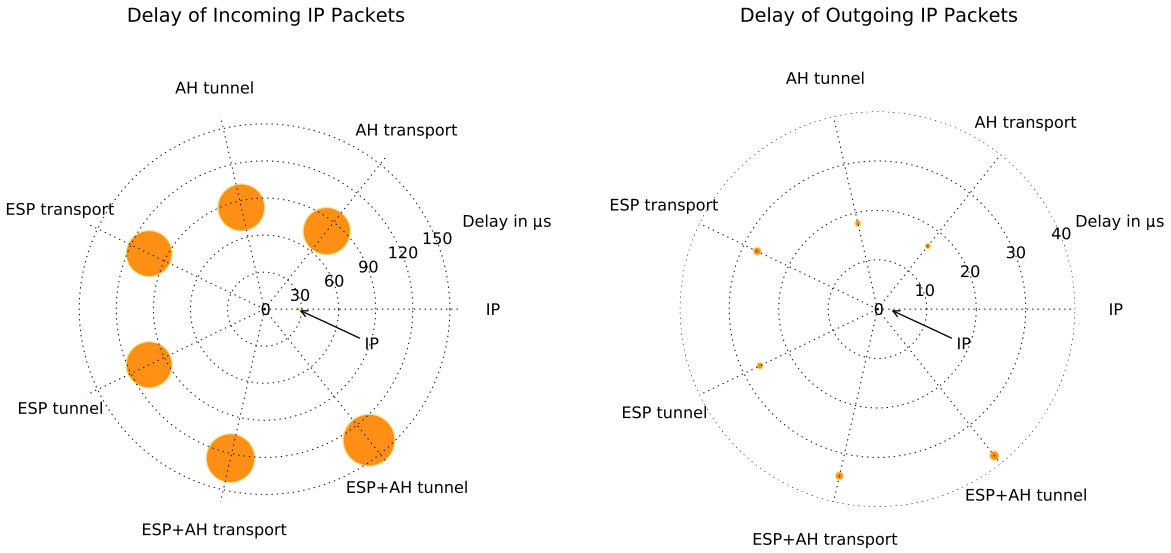

Figure 6.11: Comparison of PDV results; the circles represent the standard deviation in scale 1:10.

6.3 Clock Synchronization using PTPv2

In opposite to the previous section, which only considers the delay in the IP stack, the measurements in this section cover the precision of the clock synchronization including all system components.

A PTP stack in slave mode measures the offset from the master with the help of synchronization and delay messages. These values are fed back to the control loop of the PTP stack, which takes care of the adjustment of the clock and the clock rate. The deviation of the clocks, in terms of precision, defines the influence of the underlying stack.

All the diagrams shown in this section follow the same logic. The upper subfigure shows the offset from the slave clock to the master clock over the measurement cycle. A measurement cycle records the offset for 30 minutes. Every second a measurement is performed, which is triggered by the one Pulse Per Second (PPS) output of the two computers. The lower subfigure shows the histogram of the offset, depicted in the upper part of the diagram.

6.3.1 PTPv2 over IPv4 without Security Extension

The measurement of the offset between two synchronized nodes over IPv4 without security extension, neither native PTP solution nor IPsec is depicted in Figure 6.12. This measurement is used as a reference, since no security is included.

Figure 6.12a shows the offset over the measurement cycle, the values for the offset are residing around 147 µs. The visible oscillation reflects the adjustment of the clock rate to the measured values.

The histogram for this measurement is shown in Figure 6.12b. Most of the measurement values are between 143 µs and 148 µs.

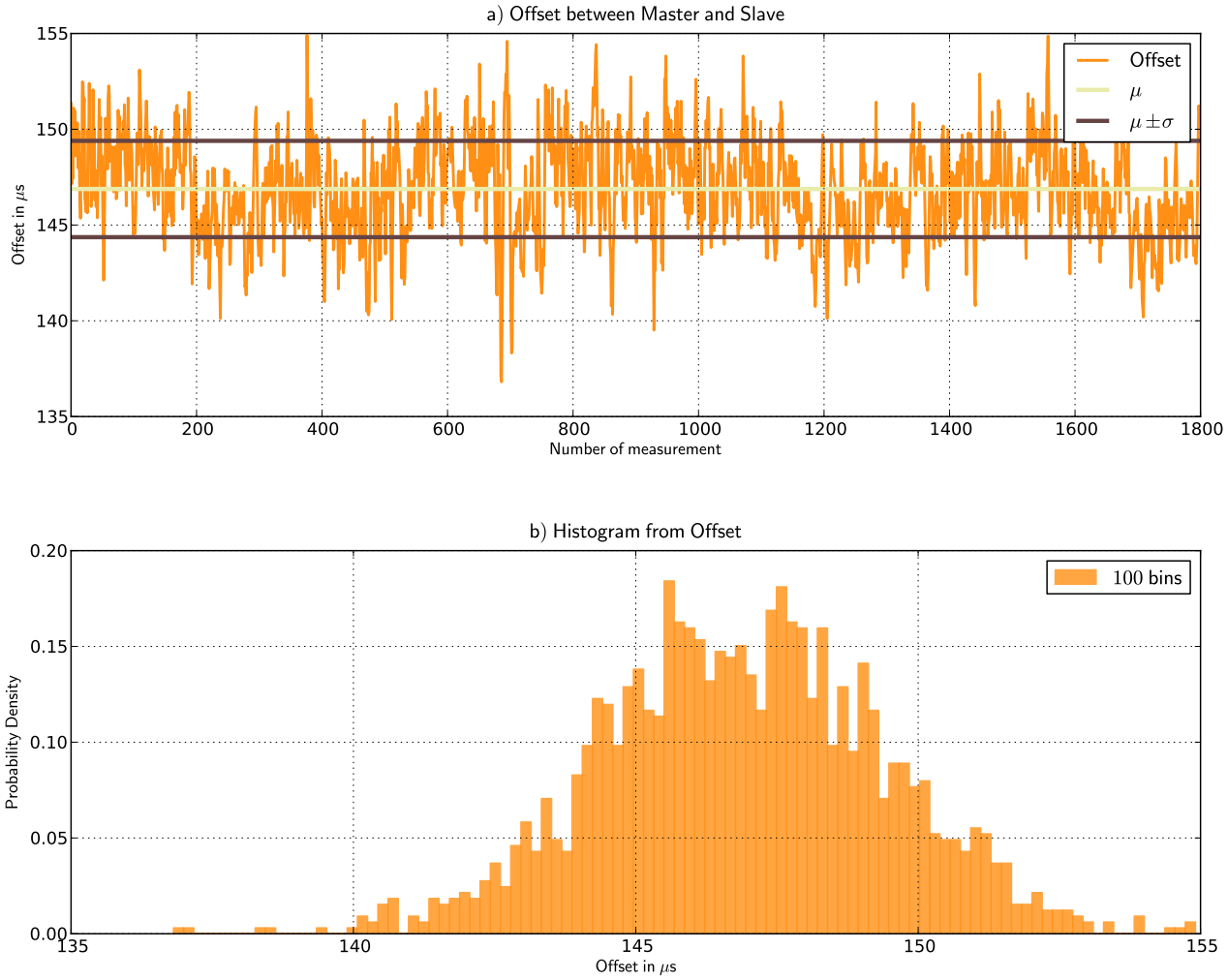

Figure 6.12: PTPv2 synchronizing over IPv4 without security extension

6.3.2 PTPv2 with Native Security Extension Annex K

The measurement of the offset between two synchronized nodes over IPv4 with Annex K, the native security extension of IEEE 1588v2, is depicted in Figure 6.12 a and b.

Figure 6.12a shows the offset over the measurement cycle, the values for the offset are residing around 237 µs. Again the oscillation reflects the adjustment of the clock rate to the measured values. Compared to the previous measurement cycle, IPv4 only, the offset has been increased by 90 µs.

The histogram for this measurement is shown in Figure 6.12b. This time the measurement values are between 236 µs and 241 µs.

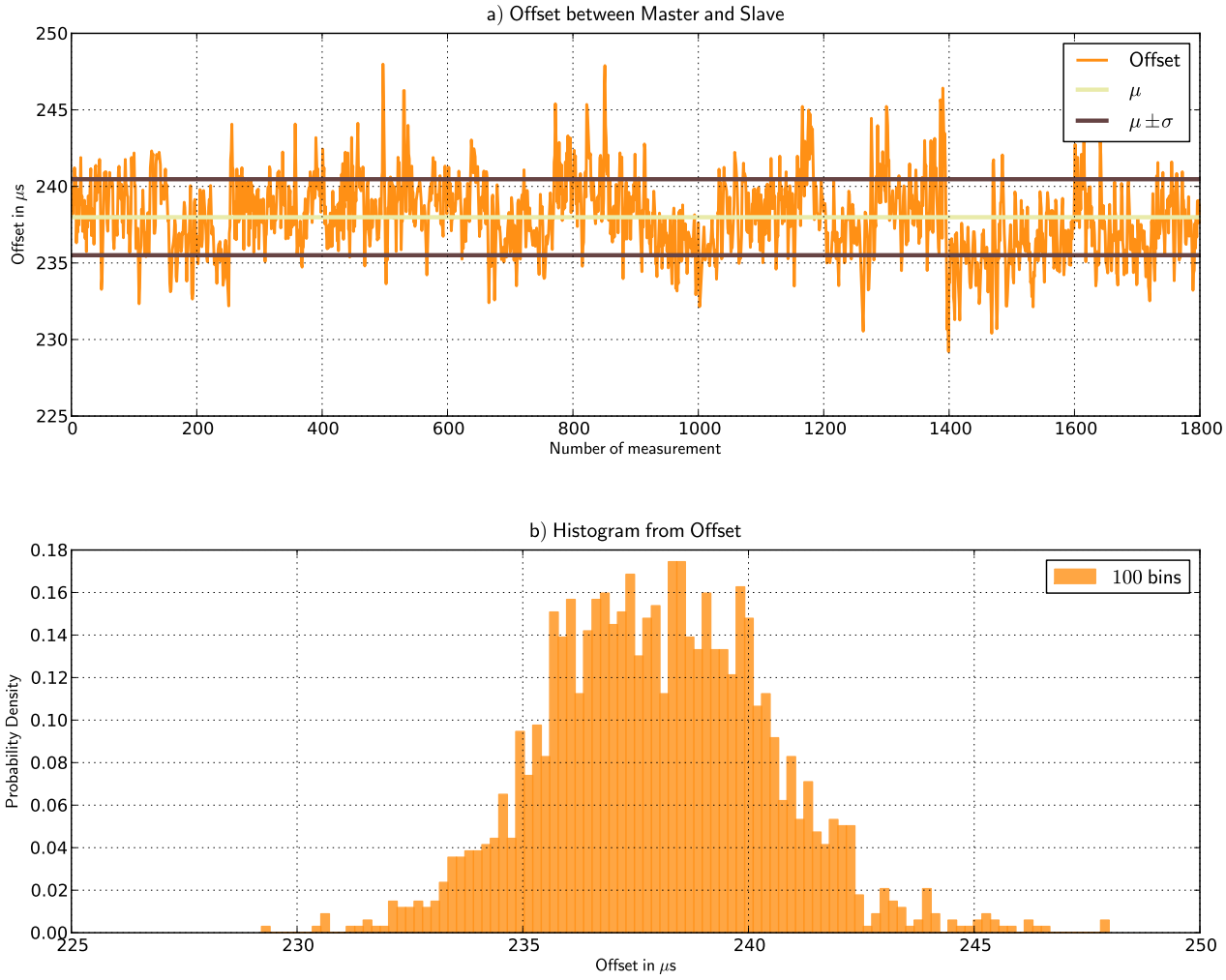

Figure 6.13: PTPv2 synchronizing over IPv4 with Annex K, the native security extension of IEEE 1588v2

6.3.3 PTPv2 with Authentication Header in Transport Mode

The measurement of the offset between two synchronized nodes over IPv4 with Authentication Header (AH) in transport mode is depicted in Figure 6.14.

Figure 6.14a shows the offset over the measurement cycle, the values for the offset are settling down a little bit under 155 µs. Again oscillations from the control loop are visible. Comparing the current measurement cycle to the previous two shows one interesting effect. The offset of IPv4 is marginally lower, about 7 µs, and for the native solution it is 80 µs higher. IPsec is processed in kernel space only and uses highly optimized cryptographic libraries. The native solution is executed completely in user mode, using the OpenSSL library[1]. Both security extension use the same calculation schemes. Therefore, the difference between the two offsets reflects the time for executing the cryptographic calculations, either in kernel space or in user space, respectively.

The histogram for this measurement is shown in Figure 6.14b. This time, most of the measurement values are located between 150 µs and 158 µs.

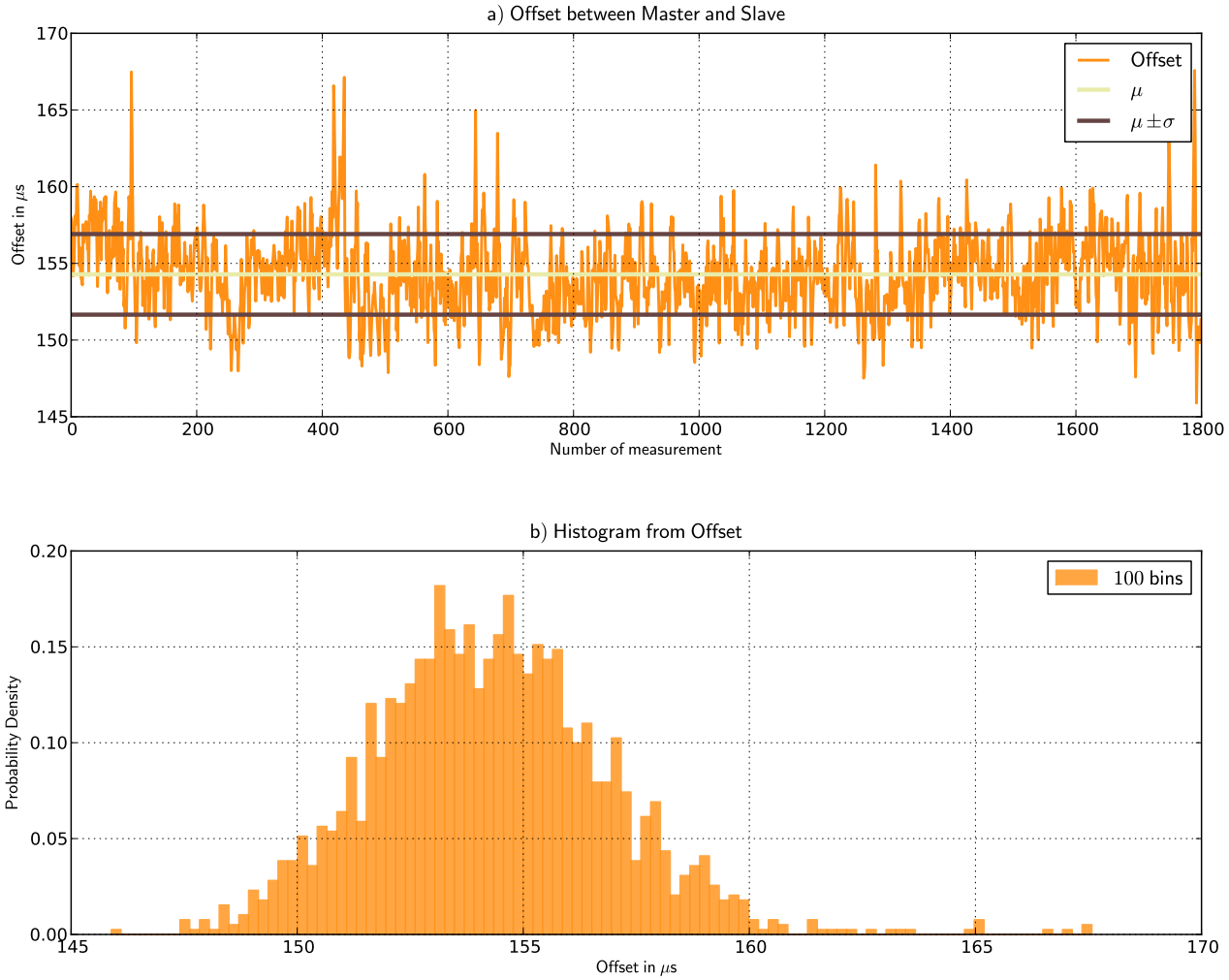

Figure 6.14: PTPv2 synchronizing over IPv4 with Authentication Header in transport mode

[1] http://www.openssl.org/

6.3.4 PTPv2 with Authentication Header in Tunnel Mode

The measurement of the offset between two synchronized nodes over IPv4 with Authentication Header in tunnel mode is depicted in Figure 6.15.

Figure 6.15a shows the offset over the measurement cycle, settling around 155 µs, similar to the previous measurement cycle, Authentication Header in transport mode. Oscillations from the control loop are visible. Compared to the previous IPsec measurements, no big difference can be spotted.

The histogram for this measurement is shown in Figure 6.15b. Similar to the previous measurement, most of the values are within a range of 10 µs. No significant visible similarity can be found between this measurement and the corresponding PDV measurement.

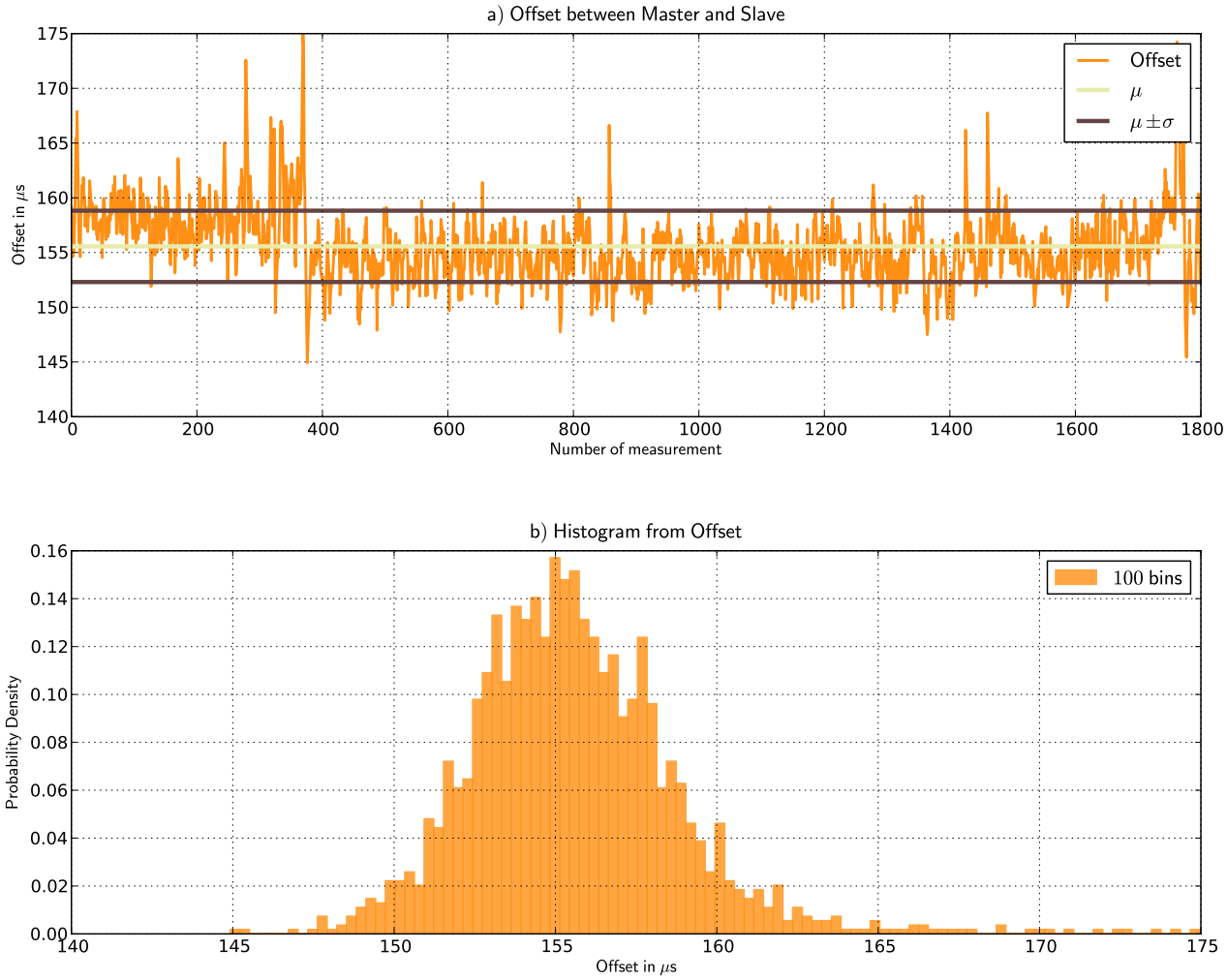

Figure 6.15: PTPv2 synchronizing over IPv4 with Authentication Header in tunnel mode

6.3.5 PTPv2 with Encapsulating Security Payload in Transport Mode

The measurement of the offset between two synchronized nodes over IPv4 with Encapsulating Security Payload in transport mode is depicted in Figure 6.16.

Figure 6.16a shows the offset over the measurement cycle. This time the offset is oscillating a little bit under 160 µs, with almost no outliers present. This time also present oscillations from the control loop. Compared to the previous measurements only, a slight increase for offset is noticeable.

The histogram for this measurement is shown in Figure 6.16b. Also this time, the width of the histogram is approximately 10 µs.

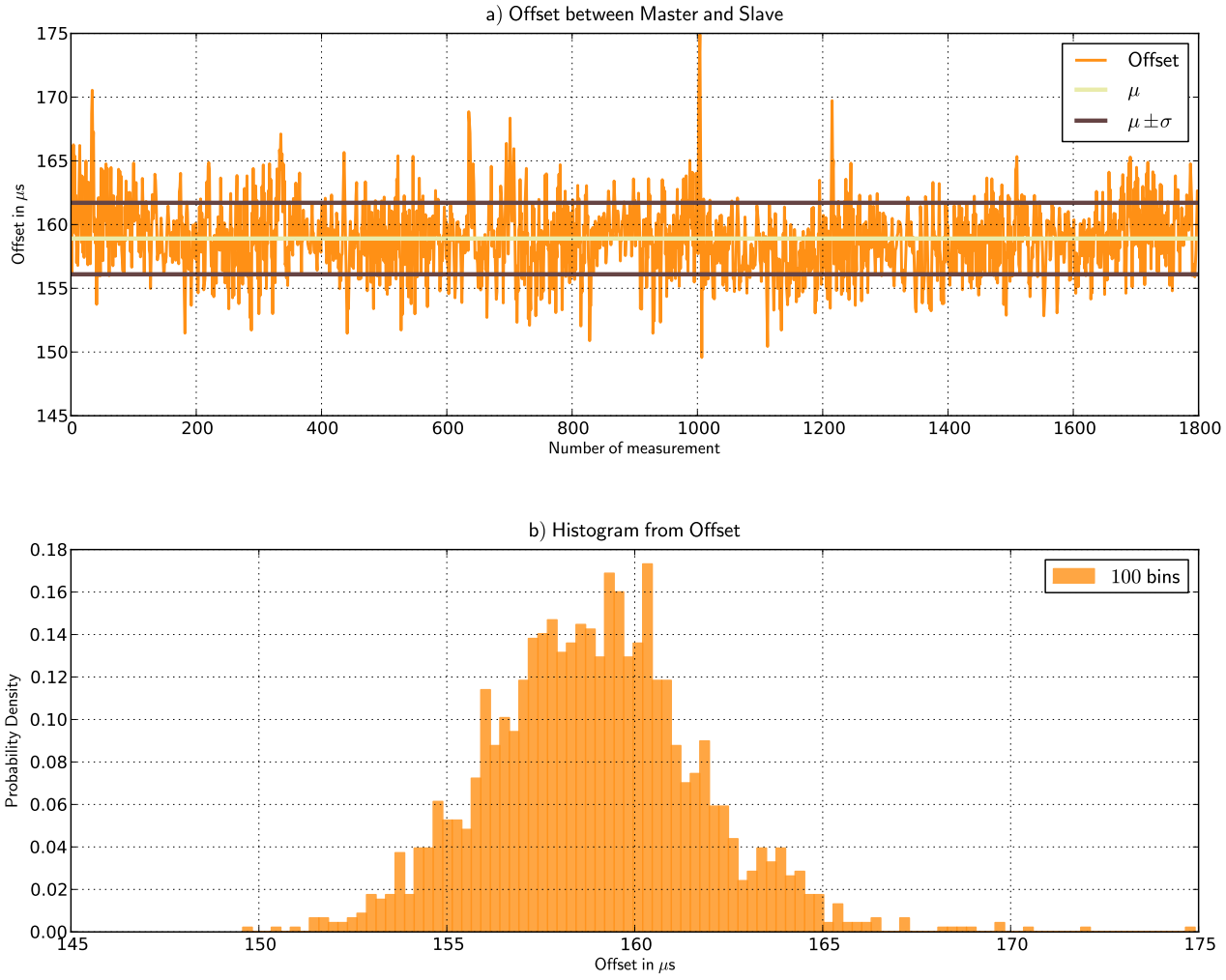

Figure 6.16: PTPv2 synchronizing over IPv4 with Encapsulating Security Payload in transport mode

6.3.6 PTPv2 with Encapsulating Security Payload in Tunnel Mode

The measurement of the offset between two synchronized nodes over IPv4 with Encapsulating Security Payload in tunnel mode is depicted in Figure 6.17.

Figure 6.17a shows the offset over the measurement cycle. Due to the control loop, the offset is oscillating, and the center is located at 155 µs. The offset is comparable to the previous measurement cycles and does not introduce an additional offset.

The histogram for this measurement is shown in Figure 6.17b. In this histogram, the width has decreased a little bit and it is nowe less than 10 µs. The matching PDV measurement has two peaks, which have a width of less than a range of 10 µs, and it is comparable to this measurement result. ESP in tunnel mode has less offset than ESP in transport mode, in contrast to the rest of the measurements, which indicates that more headers and more cryptographic calculations cause bigger offset. Repeated measurements show the same effect.

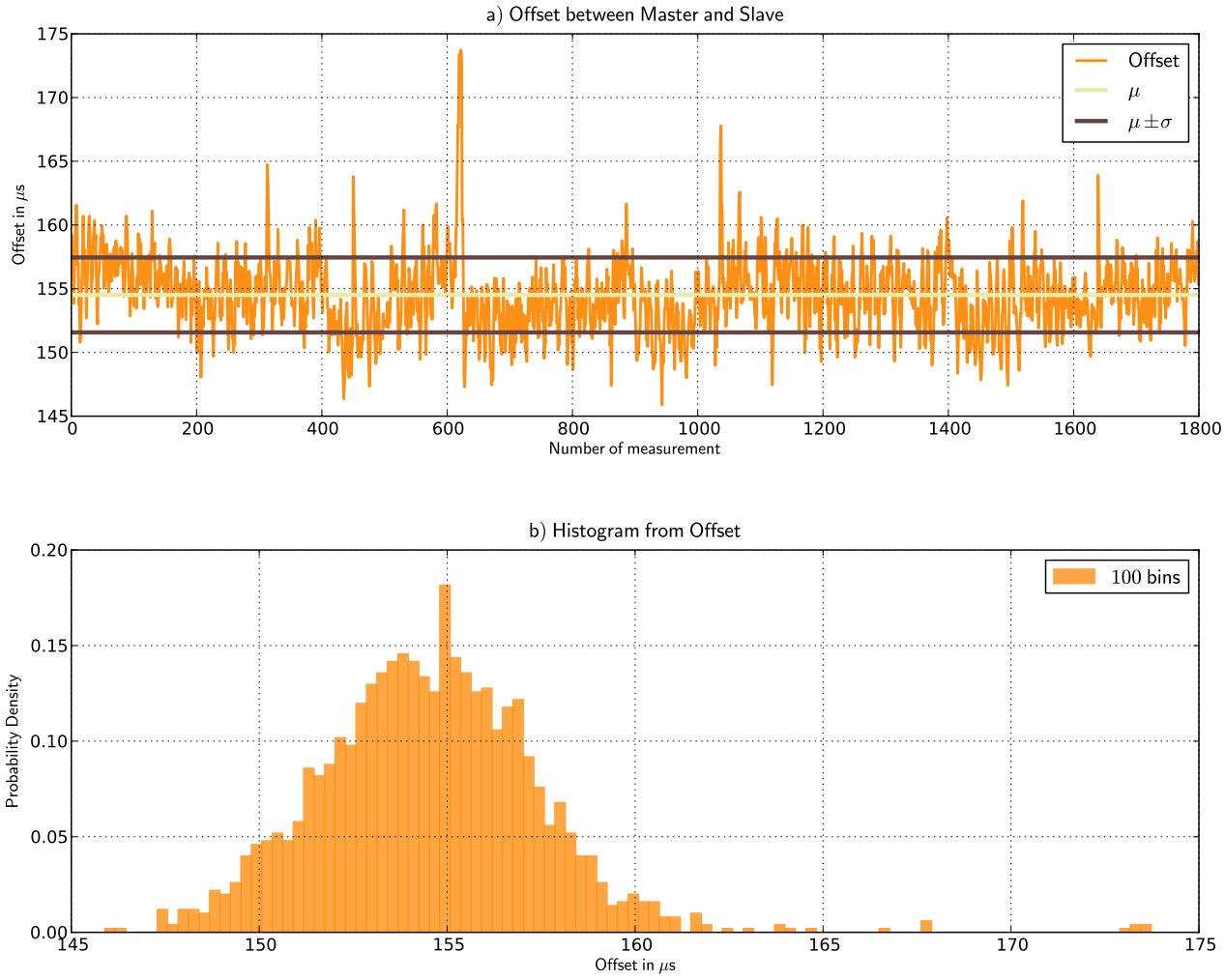

Figure 6.17: PTPv2 synchronizing over IPv4 with Encapsulating Security Payload in tunnel mode

6.3.7 PTPv2 with Encapsulating Security Payload and Authentication Header in Transport Mode

The measurement of the offset between two synchronized nodes over IPv4 with Encapsulating Security Payload and Authentication Header in transport mode is depicted in Figure 6.18.

Figure 6.18a shows the offset over the measurement cycle. This measurement is matching the previous IPsec measurement cycles, the oscillation is centered around 155 µs.

The histogram for this measurement is shown in Figure 6.18b. In this histogram, the width has decreased a little bit and it is less than 10 µs, just like in Encapsulating Security Payload in tunnel mode.

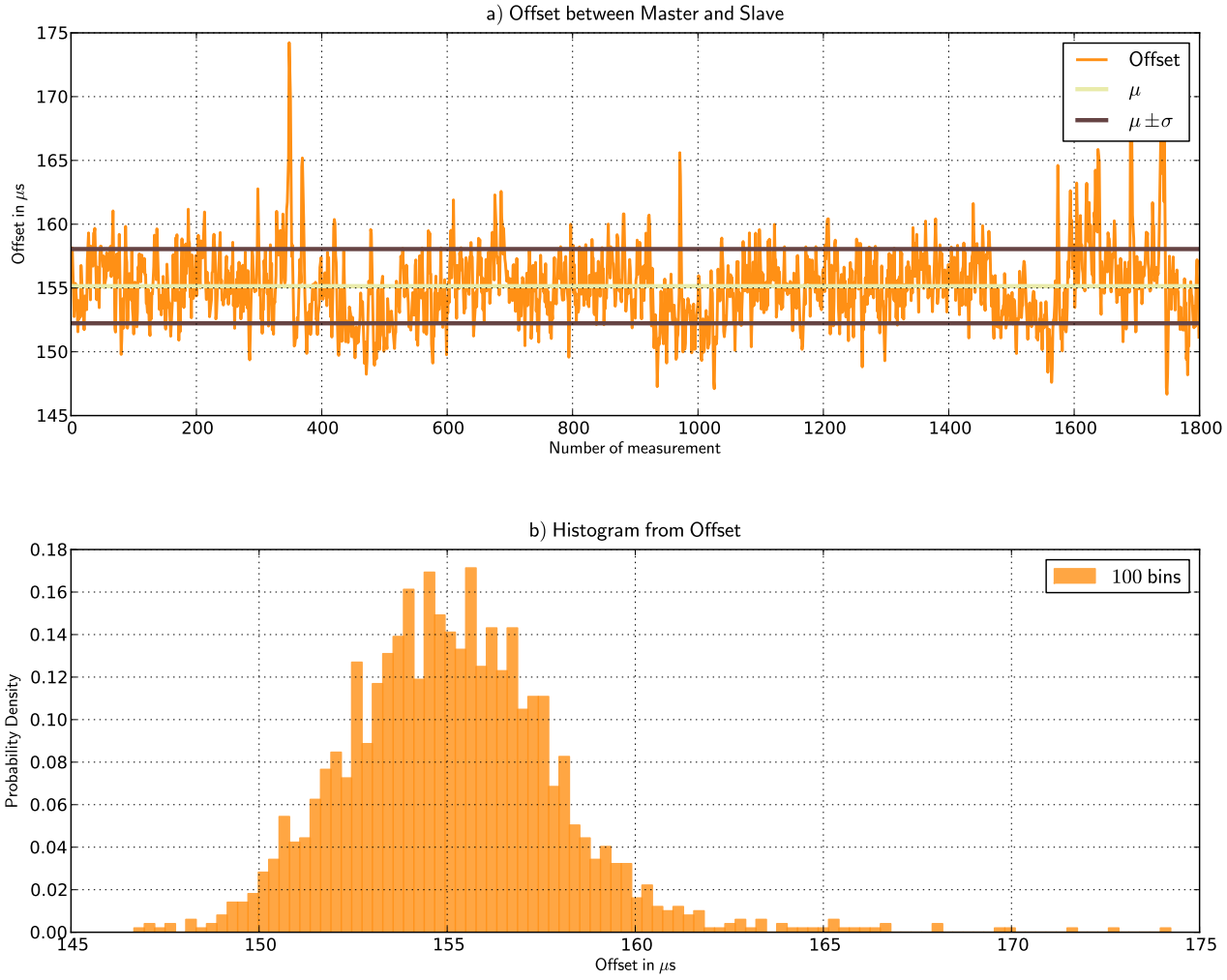

Figure 6.18: PTPv2 synchronizing over IPv4 with Encapsulating Security Payload and Authentication Header in transport mode

6.3.8 PTPv2 with Encapsulating Security Payload and Authentication Header in Tunnel Mode

The measurement of the offset between two synchronized nodes over IPv4 with Encapsulating Security Payload and Authentication Header in transport mode is depicted in Figure 6.19.

Figure 6.19a shows the offset over the measurement cycle. This measurement is matching the previous measurement cycles, the oscillation is centered around 155 µs.

The histogram for this measurement is shown in Figure 6.19b. Similar to the previous measurement cycles, the width has decreased a little bit and it is less than 10 µs, just like in Encapsulating Security Payload in tunnel mode.

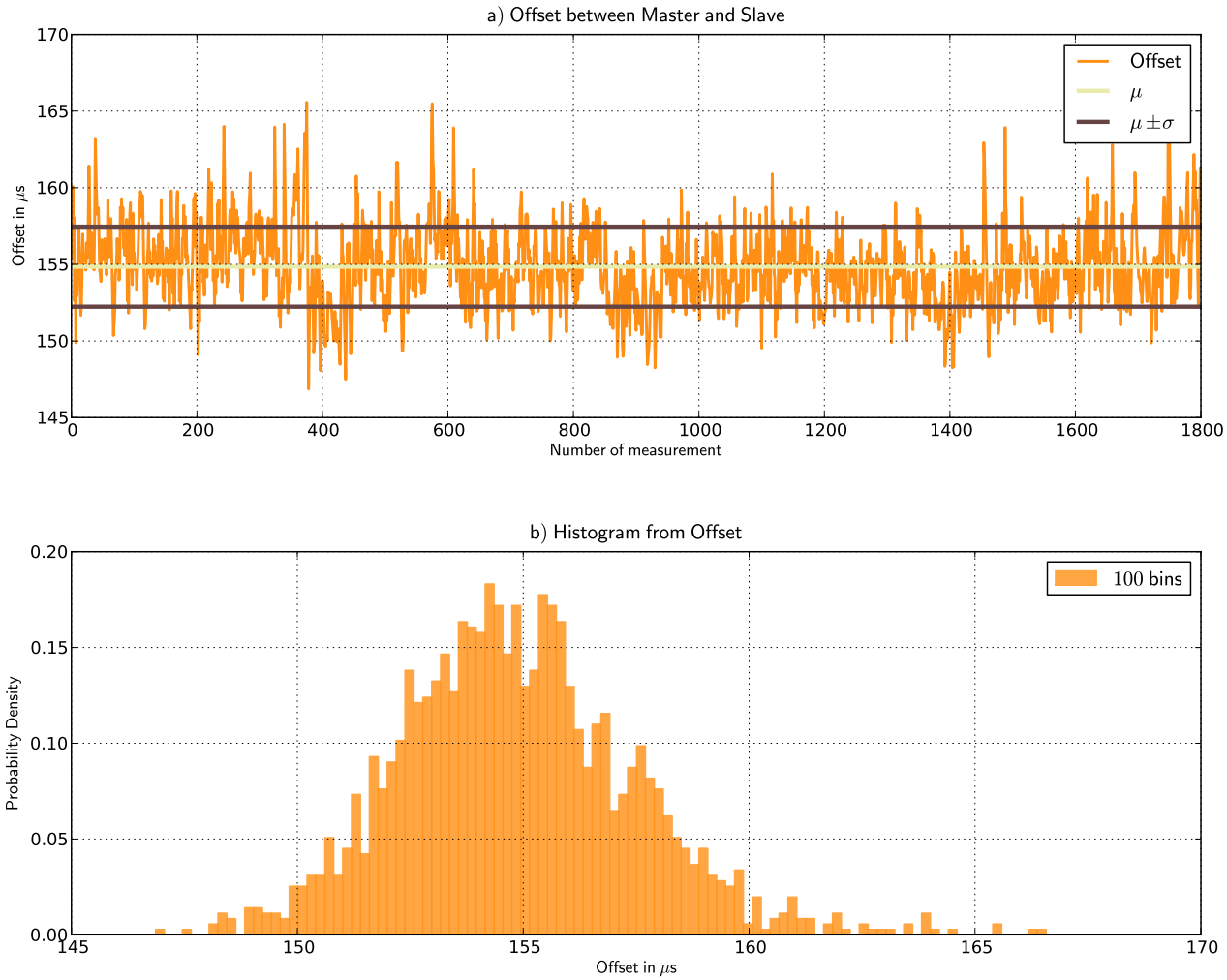

Figure 6.19: PTPv2 synchronizing over IPv4 with Encapsulating Security Payload and Authentication Header in tunnel mode

6.3.9 Summary of the Synchronization Measurements

The aggregation of the measurement cycle for the synchronization measurements can be found in Table 6.3.

IP configuration	Offset ($\mu \pm \sigma$)
IPv4	147.11 µs ±2.62 µs
IPv4 with Annex K	237.98 µs ±2.49 µs
IPv4 AH Transport	154.29 µs ±2.62 µs
IPv4 AH Tunnel	156.14 µs ±3.57 µs
IPv4 ESP Transport	159.37 µs ±3.06 µs
IPv4 ESP Tunnel	154.76 µs ±3.05 µs
IPv4 ESP and AH Transport	155.38 µs ±3.02 µs
IPv4 ESP and AH Tunnel	154.91 µs ±2.63 µs

Table 6.3: Roundup of the synchronization results

The table shows the absolute value of the offset and the standard deviation between a master and a slave. The lowest offset occurs in the IPv4 case. All security enhancements increase the offset, however the standard deviation is only increased marginally from 2.49 µs to 3.57 µs.

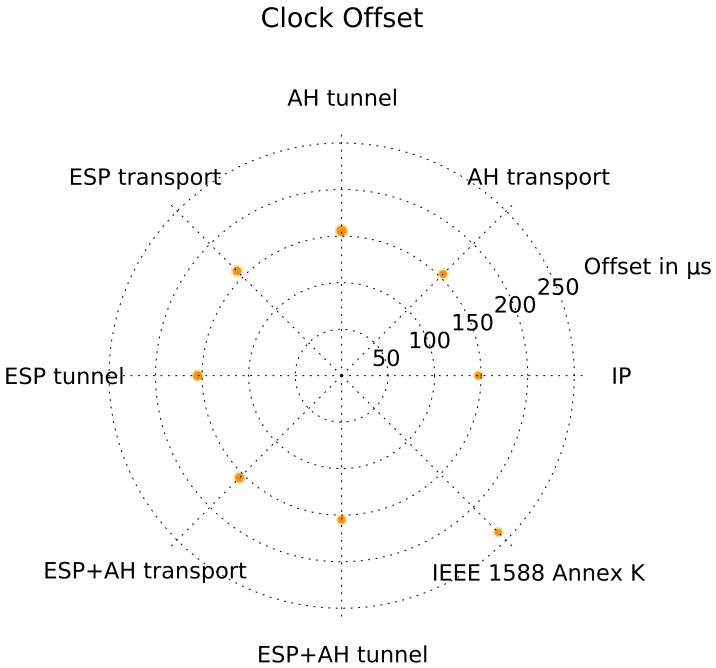

Figure 6.20: Comparison of the synchronization results; the circles represent the standard deviation in scale 1:10.

The native security extension of PTPv2 shows the highest increase of the offset. In contrast to the IPsec modes, which have an offset much closer to the reference scenario, depending on the

mode the offset is 7 µs to 12 µs.

From the architectural point of view the big difference between the native solution and the IPsec approach is the context in which they run. IPsec runs in kernel space while the native security runs in user space. Both solutions use different cryptographic libraries. Annex K uses the OpenSSL cryptographic library and the kernel has its own cryptographic API. The different cryptographic functions execute in different contexts and have different implementations, therefore the cryptographic functions used in the user space approach have a considerably longer execution time. Nevertheless, the standard deviations of the different modes are very close together. Therefore both implementations have the same quality in the context of clock synchronization, only differing in the absolute value of the offset that is compensated by the control algorithm anyway.

Figure 6.20 shows the visualization of the clock synchronization results. All measurements share almost the same offset, the only exception is the native PTPv2 security. Due to the different cryptographic libraries and the execution in user space. The standard deviation is represented with the area of the circles and it is almost identical for all measurements.

7 Conclusion and Perspectives

The results and insights gained during this work are summarized in the first part of this chapter. Furthermore, an outlook is given in the second part of this chapter, which details topics that are worthy to pursue and to extend.

7.1 Conclusion

The analysis of the security of Annex K from the IEEE 1588v2 standard is done in this work. It shows that the measures, which are taken in the protocol, are partly flawed. For the deployment in a productive environment the options, which are available in the security extension, have to be used with caution, due to the fact that the protocol address is not included in the calculation of the ICV. The identified flaw allows to launch a man-in-the-middle attack on any PTP node present in the network. Such an attack can have a devastating effect on the clock synchronization and all services, based on correct timing information.

The security extension in IEEE 1588v2 also introdcues a challenge-response procedure for authentication. In contrast to non-secure networks, the nodes have to establish Security Associations (SA) before any clock synchronization information is exchanged. This start-up phase introduces three additional messages to authenticate two nodes. This additional traffic might be negligible in smaller networks. However, in large networks with several hundreds of nodes this initialization phase can lead to a serious degradation in performance, when all nodes are turned on at once [TGHC07].

After the security analysis, the security extension of the PTP was implemented. The implementation is completely encapsulated and can be ported to stacks from different vendors. Due to the fact that this implementation has to work on resource limited devices, the design favors an efficient design. Most of the implementation is modular, which makes it possible to change parts and test different implementations depending on the needs of a platform.

After the implementation of the security extension, the focus was put on the verification and validation of the secure clock synchronization stack. Before testing the application for the correct behavior, static code analysis was performed to find possible candidates for non clean programmed parts and potential flaws leading in particular to attack threats. Based on the standard, several tests have been developed to ensure the correct behavior of the software. These tests can be used to validate existing implementations of the security extension.

Additionally, real measurements have been taken, giving insight into the behavior of secure clock synchronization networks. Until now, the effects of security schemes could only be guessed, but no scientific analysis was available. The analysis focuses on Annex K of IEEE 1588v2 and IPsec. Both security schemes are evaluated with regard to the Packet Delay Variation (PDV) and the impact on the clock synchronization. These values are compared to IPv4 without any activated security schemes.

The PDV for IPv4 shows that the incoming and outgoing delay is asymmetrical, with a minimum input delay of 80.73 µs ±17.79 µs and a maximum input delay of 135.58 µs ±19.52 µs, for IPsec. The output of the security has a minimum delay of 16.33 µs ±2.18 µs and a maximum delay of 37.91 µs ±3.59 µs, for IPsec. The different modes provided by IPsec show a significant change for these values. The increase of the delay is correlated to the applied security, more headers or increased demand for encryption and authentication results in higher values for input and output delay. The highest values for the delay can be observed when ESP and AH are applied together.

The second set of measurements puts the focus on the clock synchronization between a master and a slave node. Both clocks run PTP for synchronization, facilitating the different security schemes already used in the first set of the measurements. The resulting offset between the two clocks for IP and IPsec is very similar. The only exception in this group is the implementation of Annex K. However, this change is only reflected in the offset. The standard deviation for all security modes is very close to each other, in range of 2.6 µs to 3.6 µs.

This work shows that the influence of security on high-precision clock synchronization in software is negligible to the deviation produced by other components in the operating system, such as schedulers and harddisk read and write operations.

7.2 Perspectives

The current solution is implemented completely in software. The next step to achieve better clock synchronization would be to implement the complete system in hardware. The design in hardware offers new challenges, such as secure on-the-fly timestamping. The implementation needs to deal with the more stringent hardware demands for on-the-fly timestamping, which draws the timestamp in hardware, adds the information to the packet, and recalculates all necessary checksums. An obvious obstacle, which has to be solved on the way to a working prototype, is the combination of hardware timestamp and authentication of a PTP message. These are two competing processes, both having the demand to be the last operation in the packet processing. The timestamp needs to be taken as late as possible to offer the best accuracy and the authentication of the packet can only be applied after the packet is entirely available.

The algorithms used for the IPsec measurements in this diploma thesis closely followed the proposed algorithms provided in IEEE 1588v2. An evaluation of different calculation schemes can provide information about the computational effort needed for a specific algorithm and how this impacts the clock synchronization. IPsec supports a broad range of encryption and authentication algorithms. A combination of schemes can be used to find a solution, which yields to a minimum of system load, or a solution, which has the lowest impact on the clock synchronization itself.

An aspect, which is also of interest, is the speed of a network. At first glance, higher network speeds seem to be preferable to achieve higher accuracy. This argument is not a valid assumption even for networks without security extensions. Due to the different approaches taken by the physical layer,

a hardware solution yields different results for 10 MBit/s, 100 MBit/s, or 1000 MBit/s networks, where the highest speed not necessarily reflects the highest accuracy [Los10]. When it comes to software approaches, there might be little to no difference between implementations with and without security, similar to the results gathered in this diploma thesis. A hardware support for security has completely different timing constraints for the different network speeds, according to [THS10].

Currently, the attention is paid on the master and slave devices in the network. A look beyond these elements needs to be done to include all network elements participating in the communication. For practical applications, an evaluation of a secure switch architecture with support for hardware timestamping has to be conducted. IPsec, which is used in this diploma thesis, offers end-to-end protection for an IP packet. A switch that manipulates a message automatically destroys the integrity of the packet and the receiver will drop such a packet. To be able to change parts of the packet and generate valid output, the switch would have to have keys, which are shared between two communicating nodes. However, this defeats the purpose of the end-to-end security. Further, investigations can be done on the field of security schemes, which incorporate all network elements that join communication.

To be able to use a security scheme in a network, keys are needed for authentication, and also for encryption. This task can become cumbersome when the network consists of several hundred nodes and the keys have to be renewed in a periodic interval, ranging from days to weeks. Therefore, a suitable key management is needed, that fulfills these requirements. The conditions for such a scheme are diverse and a solution suitable for all applications might not be feasible. Research in the area of key management for high numbers of resource limited nodes can be conducted.

Literature

[Bri10] Briscoe, B.: *Tunnelling of Explicit Congestion Notification.* RFC 6040 (Proposed Standard). `http://www.ietf.org/rfc/rfc6040.txt`. Version: November 2010

[DH98] Deering, S. ; Hinden, R.: *Internet Protocol, Version 6 (IPv6) Specification.* RFC 2460 (Draft Standard). `http://www.ietf.org/rfc/rfc2460.txt`. Version: December 1998 (Request for Comments). – Updated by RFCs 5095, 5722, 5871

[DR02] Daemen, Joan ; Rijmen, Vincent: *The Design of Rijndael: AES - The Advanced Encryption Standard.* Berlin, Heidelberg, New York : Springer Verlag, 2002. – ISBN 3–540–42580–2

[GTS06] Gaderer, Georg ; Treytl, Albert ; Sauter, Thilo: Security Aspects for IEEE 1588 based Clock Synchronization Protocols. In: Cena, Gianluca (Hrsg.) ; Vasques, Francisco (Hrsg.) ; IEIIT–CNR and IEEE Industrial Electronics Society and ANIPLA (Veranst.): *2006 IEEE International Workshop on Factory Communication Systems Proceedings.* Turin, Italy, June 2006, S. 247–250. – IEEE Catalog Number: 06TH8884

[HM10] Haberman, B. ; Mills, D.: *Network Time Protocol Version 4: Autokey Specification.* RFC 5906 (Informational). `http://www.ietf.org/rfc/rfc5906.txt`. Version: June 2010

[IEE08] IEEE: IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems. In: *IEEE Std 1588-2008 (Revision of IEEE Std 1588-2002)* (2008), 24, S. c1 –269. `http://dx.doi.org/10.1109/IEEESTD.2008.4579760`. – DOI 10.1109/IEEESTD.2008.4579760

[Ken05a] Kent, S.: *IP Authentication Header.* RFC 4302 (Proposed Standard). `http://www.ietf.org/rfc/rfc4302.txt`. Version: December 2005

[Ken05b] Kent, S.: *IP Encapsulating Security Payload (ESP).* RFC 4303 (Proposed Standard). `http://www.ietf.org/rfc/rfc4303.txt`. Version: December 2005

[KHNE10] Kaufman, C. ; Hoffman, P. ; Nir, Y. ; Eronen, P.: *Internet Key Exchange Protocol Version 2 (IKEv2).* RFC 5996 (Proposed Standard). `http://www.ietf.org/rfc/rfc5996.txt`. Version: September 2010 (Request for Comments). – Updated by RFC 5998

[KS05] Kent, S. ; Seo, K.: *Security Architecture for the Internet Protocol*. RFC 4301 (Proposed Standard). `http://www.ietf.org/rfc/rfc4301.txt`. Version: December 2005 (Request for Comments). – Updated by RFC 6040

[Los10] Loschmidt, Patrick: *On Enhanced Clock Synchronization Performance Through Dedicated Ethernet Hardware Support*. Viktor Kaplan Straße 2, 2700 Wiener Neustadt, Diss., January 2010

[Man07] Manral, V.: *Cryptographic Algorithm Implementation Requirements for Encapsulating Security Payload (ESP) and Authentication Header (AH)*. RFC 4835 (Proposed Standard). `http://www.ietf.org/rfc/rfc4835.txt`. Version: April 2007

[Mil06] Mills, David L.: *Computer Network Time Synchronization: The Network Time Protocol*. Boca Raton, FL, USA : CRC Press, Inc., 2006. – ISBN 0849358051

[MMBK10] Mills, D. ; Martin, J. ; Burbank, J. ; Kasch, W.: *Network Time Protocol Version 4: Protocol and Algorithms Specification*. RFC 5905 (Proposed Standard). `http://www.ietf.org/rfc/rfc5905.txt`. Version: June 2010

[MO85] Marzullo, Keith ; Owicki, Susan: Maintaining the time in a distributed system. In: *SIGOPS Oper. Syst. Rev.* 19 (1985), July, 44–54. `http://dx.doi.org/http://doi.acm.org/10.1145/850776.850780`. – DOI http://doi.acm.org/10.1145/850776.850780. – ISSN 0163–5980

[MSST98] Maughan, D. ; Schertler, M. ; Schneider, M. ; Turner, J.: *Internet Security Association and Key Management Protocol (ISAKMP)*. RFC 2408 (Proposed Standard). `http://www.ietf.org/rfc/rfc2408.txt`. Version: November 1998

[NIS02] NIST: *Secure Hash Signature Standard (SHS) (FIPS PUB 180-2)*. 2002

[Riv92] Rivest, R.: *The MD5 Message-Digest Algorithm*. RFC 1321 (Informational). `http://www.ietf.org/rfc/rfc1321.txt`. Version: April 1992

[Sch95] Schneier, Bruce: *Applied cryptography (2nd ed.): protocols, algorithms, and source code in C*. New York, NY, USA : John Wiley & Sons, Inc., 1995. – ISBN 0–471–11709–9

[SKTV06] Sakane, S. ; Kamada, K. ; Thomas, M. ; Vilhuber, J.: *Kerberized Internet Negotiation of Keys (KINK)*. RFC 4430 (Proposed Standard). `http://www.ietf.org/rfc/rfc4430.txt`. Version: March 2006

[Tel04] Telecommunications and Timing Group (TTG) of the Range Commanders Council (RCC): IRIG serial time code formats. In: *IRIG Standard 200-04 (update of IRIG Standard 200-98)* (2004), September. `https://wsmrc2vger.wsmr.army.mil/rcc/PUBS/pubs.htm`

[TGHC07] Treytl, Albert ; Gaderer, Georg ; Hirschler, Bernd ; Cohen, Ron: Traps and Pitfalls in Secure Clock Synchronization. In: Gaderer, Georg (Hrsg.) ; Lee, Kang (Hrsg.): *Proceedings of 2007 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control and Communication*. Vienna : Austrian Academie of Sciences, October 2007, S. 18–24

[TH08] Treytl, Albert ; Hirschler, Bernd: Practical Application of 1588 Security. In: Treytl, Albert (Hrsg.) ; Ya-Shian, Li-Baboud (Hrsg.): *Proceedings of 2008 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control and Communication*, 2008, S. 37–43

[TH09] Treytl, Albert ; Hirschler, Bernd: Security Flaws and Workarounds for IEEE 1588 (Transparent) Clocks. In: *Proceedings of 2009 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control and Communication*. Brescia, October 2009, S. 103–108

[TH10] Treytl, Albert ; Hirschler, Bernd: Securing IEEE 1588 by IPsec Tunnels - An Analysis. New Hampshire, USA : Copyright 2010 by the Institute of Electrical and Electronics Engineers, October 2010, S. 83–90

[THS10] Treytl, Albert ; Hirschler, Bernd ; Sauter, Thilo: Secure Tunneling of High Precision Clock Synchronisation Protocols and other time-stamped data. In: Willig, Julián Proenza A. (Hrsg.): *2010 IEEE International Workshop on Factory Communication Systems Proceedings*. Nancy, France : Copyright 2010 by the Institute of Electrical and Electronics Engineers, May 2010, S. 303–313

Annex "K" – Cryptographic Acknowledgment

The origin of the word eavesdropper had nothing to do with a vicious act of a person. The first meaning of this word referred to water that fell from the eaves of a house to the ground. Finally, an eavesdropper was the name for a person who was standing within the eavesdrop of a house to listen to conversations inside. After some time, the meaning of the word changed to the current description, to listen secretly to private conversations.

The riddle consists of a string which includes the people I personally thank for their support. The reason for the encryption is that they are very shy and I want to protect their privacy.

The encryption used in the riddle is AES-256 with salt. To break a 256 bit key by brute force, 2^{256} ($1.15792089 \times 10^{77}$) trys are needed. A device capable of checking 10^{18} keys per second would need $3x10^{51}$ years to test for all possible keys.

AES-256 cyphertext:

```
U2FsdGVkX18YZvw6uh0H4Z83gL6q8dEwyPd9dCA5IIp689oaQtn8N8i8j2EZ11ge
```

First hint:

```
All I can see are the pictures from my measurement.
```

Second hint:

```
You have to find the key!
```