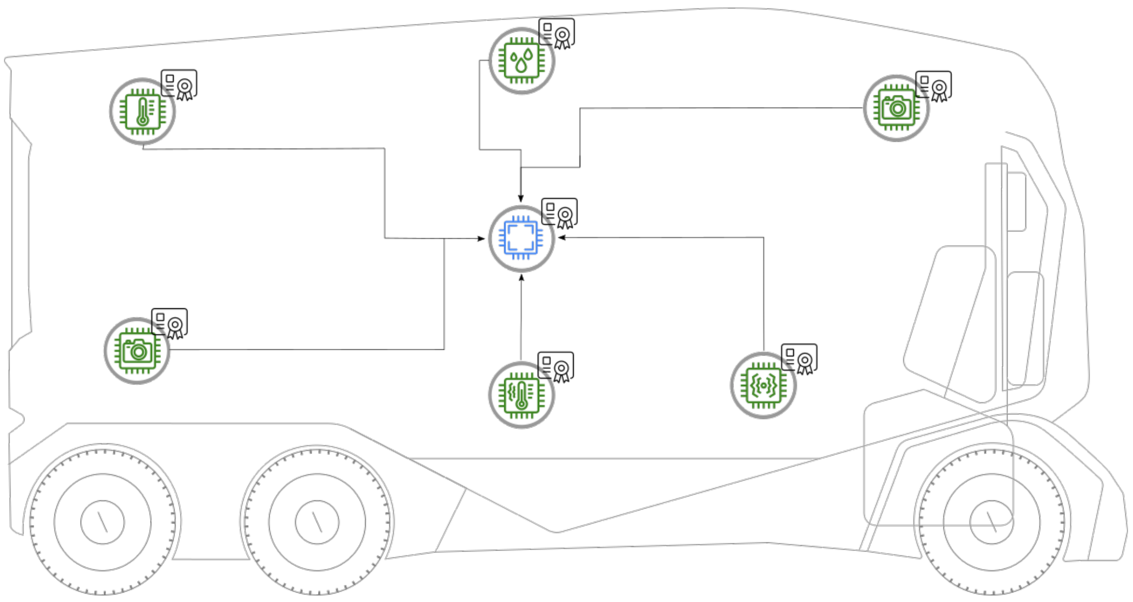

Zero Trust in Autonomous Vehicle Networks Utilizing Automotive Ethernet

John Blåberg Kristoffersson

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CHALMERS UNIVERSITY OF TECHNOLOGY
UNINVERSITY OF GOTHENBURG
Gothenburg, Sweden 2022

Master's thesis 2022

Zero Trust in Autonomous Vehicle Networks Utilizing Automotive Ethernet

John Blåberg Kristoffersson

UNIVERSITY OF GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2022

Zero Trust in Autonomous Vehicle Networks Utilizing Automotive Ethernet
JOHN BLÅBERG KRISTOFFERSSON

Supervisor: Andrei Sabelfeld, Department of Computer Science and Engineering
Examiner: Tomas Olovsson, Department of Computer Science and Engineering

Master's Thesis 2022
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2022

Abstract

If fully autonomous vehicles are going to become a reality, there has to be a significant investment in proper security measures. One of the measures that should be taken is to move the internal network security model away from perimeter security to a zero trust model. With a perimeter security model, if a malicious actor were able to connect to the in-vehicle network, they would be able to send arbitrary data to the different ECUs. The zero trust model requires continuous authentication of all devices, removing any chance for a malicious third party to connect to the network. This has not been possible due to the low bandwidth of legacy bus protocols, but with the introduction of automotive Ethernet, the bandwidth is no longer a limitation.
This thesis evaluates the viability of a zero trust network architecture in the internal network of an autonomous vehicle. This includes evaluating the performance impact of two different security protocols for guaranteeing message integrity, IPsec and MACsec, and the performance impact of retrofitting security protocol support with "Bump-in-the-Wire" devices. Lastly, a design for implementing key distribution and authentication is presented. The work found that adding message integrity did not substantially increase latency but did take up many CPU cycles. Thus, cryptographic hardware acceleration might be necessary to make a zero trust environment viable in a production setting.

Keywords: automotive security, MACsec, IPsec, automotive ethernet, zero trust

Acknowledgements

I would like to thank Oscar Söderlund for giving me the opportunity to work on this project, as well as for the continuous support throughout the process. It has been invaluable. I would also like to thank my supervisor at Chalmers, Andrei Sabelfeld, for all his great input on the work and report.

John Blåberg Kristoffersson, Gothenburg, 2022 May

List of Acronyms

Below is the list of acronyms that have been used throughout this thesis listed in alphabetical order:

AE	Automotive Ethernet
AES	Advanced Encryption Standard
ARP	Address Resolution Protocol
AV	Autonomous Vehicle
CTR	Counter Mode
DH	Diffie-Hellman Key exchange
DHCP	Dynamic Host Resolution Protocol
DoS	Denial of Service
HSM	Hardware Security Model
IKEv2	Internet Key Exchange Version Two
IVN	In-Vehicle Network
IP	Internet Protocol
LACP	Link Aggregation Control Protocol
LLDP	Link Layer Discovery Protocol
MKA	MACsec Key Agreement
OEM	Original Equipment Manufacturer
VPN	Virtual Private Network
SA	Secure Association

Contents

List of Figures

List of Tables

1 Introduction

In the past, vehicle manufacturers who wanted to ensure their passenger's safety did so by investing in the development of better airbags, new seatbelts, and more efficient crumple zones. However, with the last decade's influx of digital innovations, the driver has been progressively detached from the physical vehicle. One example of this trend is the autonomous vehicle, which operates independently from a human driver. Some tech companies, such as Swedish Einride, have taken this one step further by removing the cockpit altogether. This shift in driving responsibility can revolutionize how we transport goods and people. However, while offering considerable customer value, it also exposes the vehicle to malicious actors in a way that has not been present before.

Hacking of cars by security researchers has made headlines during the last few years, some notable being the hack of a Jeep Cherokee that forced a recall of 1,4 million vehicles and researcher Samy Kamkar's OwnStar gadget [1][2]. This has made concerns about cybersecurity real, forcing new regulations to be developed. Last year, the National Highway Traffic Safety Administration updated their Cybersecurity Best Practices for the Safety of Modern Vehicles document [3], and UNECE WP.29 regulation on cybersecurity and software updates are on the horizon [4].

One way to strengthen the security of the vehicle is to move the internal network from relying on a perimeter security model to a zero trust model. The current perimeter security model assumes that all devices with a certain perimeter, the inside of the vehicle, are trusted. On the other hand, zero trust does not intrinsically trust any device on the network. Devices must instead continuously authenticate themselves to get access to the network. In an in-vehicle network, zero trust requires a security protocol that guarantees message integrity and mutual authentication - something that's been historically hard to implement due to the limited bandwidth in in-vehicle networks.

Most modern vehicles use some bus technology as the basis of the in-vehicle network. The two most popular buss technologies, CAN and FlexRay, have bellow 1 Mb/s and 10 Mb/s of bandwidth respectively [5]. This makes implementing a security protocol difficult since the network can not afford to allocate bandwidth to cover the overhead that a security protocol brings. One technology that has long been seen as a possible alternative to these network technologies is Ethernet. It has lower latency, higher bandwidth, and supports multi-point connections. The problem has been that traditional Ethernet is too noise and interference-sensitive to be used in vehicles. However, this has changed with the IEEE standardization of 802.3bw in 2016, commonly known as automotive Ethernet.

Automotive Ethernet is a physical layer standard used in the automotive sector. It defines aspects such as wiring, signal type, and cable length. Most importantly, it lets the automotive sector utilize Ethernet's speed and bandwidth. This means that the security protocols required for a zero trust architecture are no longer prohibitively expensive due to the network's limited bandwidth. Therefore, this thesis tests the viability of implementing a zero trust architecture on an in-vehicle network [6].

This work is done in collaboration with Einride. Einride is a Swedish technology company that offers electric and autonomous shipping to help their customer reduce their CO2 emission, lower operating costs, and increase fleet productivity. It uses Einride's Pod, an electric and autonomous truck, as the reference vehicle. The in-vehicle network described throughout the thesis is based on the network that can be found in the Pod, and the threat model has been created based on the characteristics of the Pod and the scenarios and environments in which it will operate.

1.1 Purpose

This thesis aims to study the viability of implementing a zero trust architecture in an automotive Ethernet-based in-vehicle network (IVN). The work looks at the performance of two different security protocols commonly used in standard Ethernet and, based on that, concludes their applicability for automotive use. It also looks at the possibility of Bump-in-the-Wire devices to secure sensors that do not support any security protocol. Lastly, it looks at a possible way of implementing and handling the authentication and key distribution to the different devices on the network.

The two protocols that are being assessed are IPsec and MACsec. These two protocols have been chosen because they secure two different layers of the networking stack. IPsec secures IP packets, and MACsec secures Ethernet traffic. Both protocols are also open standards. IPsec is part of the IPv4 suite and comprises multiple standards created by the Internet Engineering Task Force. MACsec was released as a standard by the Institute of Electrical and Electronics Engineers. The use of open standards will hopefully ease the transition to zero trust for sensor manufacturers, as they do not have to rely on or pay for any proprietary solution.

1.2 Research Questions

This section presents the three research questions that will be the focus of this thesis. The first two questions will be researched practically, while the third will have a more theoretical focus.

1.2.1 Can protocols ensuring message integrity be effectively run on software with regards to performance?

Both IPsec and MACsec have implementations present in the Linux kernel that can be used to secure a network. However, while they work in more traditional networks, are they optimized enough to be used in an automotive environment? An in-vehicle network puts a lot more emphasis on minimal latency and efficient CPU usage, especially for autonomous vehicles that do not have a driver to correct any mistakes. The thesis will aim to answer if the implementations are viable in this environment with regard to performance.

1.2.2 Can intermediate devices be used to secure unsecured devices?

Some endpoints will inevitably not support any security protocol, at least in the foreseeable future. However, the network still has to be secure. One way of securing these endpoints is by implementing a Bump-in-the-Wire that is configured with one of the protocols in Question 1.2.1. A Bump-in-the-Wire is an intermediate device that can be retrofitted onto a device to make it support a network protocol. This way, the traffic will be secured between the Bump-in-the-Wire and the rest of the network. This question will look at the viability of implementing these devices or if they will add a prohibitable amount of latency to the network.

1.2.3 How should keys be generated, distributed, and stored in an in-vehicle network?

Both IPsec and MACsec have companion standards for handling the key exchange between ECUs: Internet Key Exchange Version Two (IKEv2) and MACsec Key Agreement (MKA), respectively. While these protocols establish how the key agreement between ECUs is handled, how the initial key is generated and inserted into the ECUs will be discussed.

1.3 Delimitations

To be able to answer the research questions within the given time frame, some delimitations have to be specified:

- **No testing is performed on a real vehicle:** The testing will be done exclusively on a testbench designed to simulate the environment of the sensor network in an autonomous vehicle. The benefit of performing tests on a real vehicle would be marginal, as the environment is harder to control, leading to unexpected results. The thesis primarily focuses on the performance of the different security protocols, which are easier to measure in a controlled environment. Although interesting for future work, the marginal benefit of performing the tests on a real vehicle is not sufficient to warrant the time investment of these types of tests.

- **The work will only focus on Automotive Ethernet:** While other network types should also be moved to the zero trust model, this work will only focus on the network segment that relies on automotive Ethernet.

- **Only authorized personnel will work on the vehicle:** Due to the complexities of letting a third-party actor work on the vehicle and the security consideration that it entails, the work will assume that only authorized personnel will work on the vehicle. Any change to the vehicle not performed by a trusted actor will be seen as malicious.

- **Only the security of internal communication will be considered:** The scope of this work limits the zero trust architecture to the vehicle's internal network. Zero trust for Vehicle-to-Vehicle (V2V), Vehicle-to-Everything (V2E), and Remote Driving (RD) will not be considered or discussed in this thesis.

- **No testing will be done on hardware security modules:** A hardware security module (HSM) is a dedicated piece of hardware meant to offload the CPU and speed up cryptographic functions. Work has already been done on measuring the performance increase from HSMs. When arguing about a protocol's viability in the automotive sector, knowledge of the potential performance boost will come from these papers instead of testing. This delimitation has taken because of time limitations and since no HSM is available for testing.

- **The goal is only to guarantee integrity, not confidentiality:** While confidentiality is important and generally seen as one of the pillars of a zero trust architecture, it is not nearly as important as integrity in this specific scenario. If an attacker were able to sniff message traffic, the attacker would only be able to acquire sensor data, which does not contain sensitive data. However, without message integrity, the attacker can spoof messages which could cause significantly more harm. Due to this difference in severity, the thesis will only focus on ensuring message integrity.

1.4 Thesis Contributions

This project presents a way of moving the in-vehicle network of an autonomous vehicle from relying on a perimeter security model to a zero trust model. It presents the performance impact of two open standard security protocols, IPsec and MACsec. By testing these two protocols an accurate estimate of the latency increase, CPU usage, and throughput reduction that comes with implementing one of these protocols is provided.

The thesis also presents an implementation of a MACsec enabled Bump-in-the-Wire device that can be used for securing nodes in the network. The performance impact of the Bump-in-the-Wire device regarding latency, throughput, and CPU utilization is also presented.

Lastly, a scheme for handling key exchanges, authentication, and authorization are discussed. This system is aligned with zero trust architecture requirements and is general enough not to rely on the two security protocols tested in the thesis.

1.5 Disposition of Thesis

The thesis is structured as follows:

- **Chapter 1 - Introduction** This chapter introduces the thesis topic and gives some background on its importance. It also contains the research questions, scope, and purpose of the thesis.

- **Chapter 2 - Background** Provides the necessary background of the central topics of the thesis. This chapter includes theory on vehicle infrastructure, the protocols the thesis will assess, and the performance and metrics used.

- **Chapter 3 - Threat Model** This chapter presents the threat model the work has been based on, the definition of zero trust used, and some possible attacks and mitigation techniques.

- **Chapter 4 - Literature Overview** Contains information on related research and how it has influenced this work.

- **Chapter 5 - Methods** Describes the methods used for answering the research questions. This includes the performance requirements, performed tests, and the different test benches used.

- **Chapter 6 - Results** This chapter presents the results of the tests.
- **Chapter 7 - Discussion** A discussion about the results and their implications on the viability of zero trust in a vehicle network.
- **Chapter 8 - Conclusion** This chapter presents the conclusion of the thesis and some future work.
- **Appendix - A** The appendix includes some tests results that were not discussed in Chapter 6

2

Background

In this section, the required theory and concepts are introduced. The chapter is structured as follows. Section 2.1 presents the general infrastructure of a modern vehicle. Section 2.2 gives a technical explanation of the protocols that are assessed in the thesis. Section 2.3 gives a brief explanation of the companion key agreements to the protocols presented in the previous chapter. Section 2.4 gives some background on the algorithms used by the protocols. Section 2.5 explains the metrics used in the evaluation and their importance.

2.1 Vehicle Infrastructure

This section gives an overview of the infrastructure of a modern vehicle and how it relates to this thesis.

2.1.1 Electronic Control Unit

This section discuss ECUs in vehicles. Note that this thesis only relates to the network that uses automotive Ethernet. This part of the network usually only includes sensors used for autonomous driving purposes and not the types of ECUs discussed in this section. This section aims primarily to give context to how an in-vehicle network functions.

An ECU is a small device that controls one or more specific electrical systems in a vehicle. Modern cars can have over 100 ECUs embedded in them, handling a wide range of functions [7]. For instance, one ECU could control essential parts such as the steering or the engine functionality. This engine-specific controller is called the Engine Control Module (ECM). The ECM is also responsible for optimizing engine performance. Other ECUs can be used simply for comfort. For example, some ECUs control power windows and seats, while others control security features such as door locks and keyless entry. Some ECUs even control safety-critical systems such as airbags or emergency breaks [8].

Depending on where in the vehicle the ECU is located and what its primary function is, it will receive information from different parts of the vehicle. An ECU controlling the airbags will receive input from crash sensors placed around the vehicle. They could also be receiving input from the seats to know which airbags to activate and which not to. An ECM can receive input from a heat sensor, an oxygen sensor, and a throttle position sensor. Based on the information from these sensors, it could then configure the engine to do what the user wants efficiently.

2.1.2 Automotive Ethernet

Legacy vehicle bus technologies such as Controller Area Network (CAN), FlexRay, and Local Interconnect Network (LIN) generally do not have more bandwidth than 10 Mbit/s, which is significantly lower than what is needed to handle the data generated by a fully autonomous vehicle. The Society of Automotive Engineers (SAE) defines six levels of sophistication for autonomous vehicles, where level zero is no automation and level five is full automaton [9]. A vehicle is estimated to require over 30 sensors for full autonomous driving, generating upwards of 40 Gbit of data per second. Even vehicles at the lower end of the spectrum will be generating about 3 Gbit/s [10]. This exponential increase in data generation has put more importance on the amount of bandwidth available in the vehicle's internal network.

Ethernet has long been seen as a possible alternative to these network technologies. The technology has proved itself in classical wired networks as a reliable transfer medium. Furthermore, it has lower latency and higher bandwidth compared to legacy technologies, as well as support for multi-point connections. The problem has been that traditional Ethernet is too noise and interference-sensitive to be used in vehicles. However, this has changed with the IEEE standardization of 802.3bw in 2016, commonly known as automotive Ethernet. The first specification was for 100BASE-T1, a physical layer standard supporting 100 Mbit/s bandwidth. In 2020, the standard was expanded to support bandwidth up to 10Gb/s. The fundamental difference between automotive Ethernet and traditional Ethernet is that automotive Ethernet is comprised of a single unshielded twisted copper pair [11]. Due to automotive Ethernet's superior bandwidth over more traditional vehicle bus technologies, it has started to see adoption in the industry [12].

2.1.3 Autonomous Vehicle Sensors

Most of the automotive Ethernet used in an in-vehicle network is used to transport data from sensors used for autonomous driving. Compared to ECUs such as those controlling power windows and airbags, which do not produce or consume a lot of data, autonomous drive sensors require the larger bandwidth that automotive Ethernet offers. While the type of data transferred does not matter for the applicability of a zero trust architecture, it can still be helpful to know the main type of data sources. Therefore, this section will briefly introduce the three main types of sensors used for autonomous driving: Radar, Lidar, and camera.

Radio detection and ranging, or Radar, is a detection system that uses radio waves to determine objects' velocity, angle, and distance. In autonomous driving, it is used for both far and near obstacle detection [13]. Radar has the advantage over Lidars and cameras in that it is much less susceptible to lighting and weather influences [14]. Furthermore, instead of measuring velocity by calculating the difference between two readings, Radar uses the Doppler effect to calculate speed. This is important for sensor fusion, the ability to combine readings from multiple types of sensors, as it gives the velocity information as independent reading, making the fusion algorithms converge much faster. However, while Radar is more efficient than cameras and Lidar in some situations, it does not produce fine-grained data [15].

Light detection and ranging, or Lidar, uses an infrared laser beam to determine the distance between an object and the sensor [15]. The sensor sends out a beam, which is the reflected by the surroundings. The sensor can then compute the distance by calculating the difference in time between the cast and received light. By doing this millions of times per second, a 3D map of the environment can be created, accurate down to a few centimeters. This ability to "see" in 3D differentiates it from technologies such as cameras. It is also far more accurate than Radar. However, they are more susceptible to weather conditions than Radar and do not measure speed as accurately [16].

Cameras are one of the most commonly used sensors for autonomous driving purposes. They allow vehicles to visualize their surrounding, which can then be used for path planning. They are widely available, cheap, and efficient. Cameras can also be used for human and machine interaction. However, they are computationally heavy. The latest cameras produce multi-megapixel images at a rate of 30-60 frames per second, which leads to large amounts of data that has to be processed in near real-time [15, 17].

2.1.4 In-Vehicle Network (IVN)

The in-vehicle network is the internal network that connects all of the ECUs, sensors, and actuators. It is usually divided into subdomains based on the type of devices it connects and these devices' use cases within the system. For instance, one such subdomain could be the real-time control applications, devices with generally low bandwidth requirements but high real-time and quality of service requirements. These can be devices controlling the brakes and traction control, and they communicate typically over CAN [18].

The domain that is in focus in this works is the sensor part of the network. This domain comprises the different sensors that is used for autonomous driving, see Section 2.1.3, and connects them to a high performance computer that contains the autonomous drive models. The network is set up in a star topology, where the hub is the high performance computer, and connected with automotive Ethernet. By using this topology the network can be effectively partitioned so that the sensors can not communicate with each other, instead only having a connection with the main hub. A visualization fo the sensor part of the network can be seen in Figure 2.1.

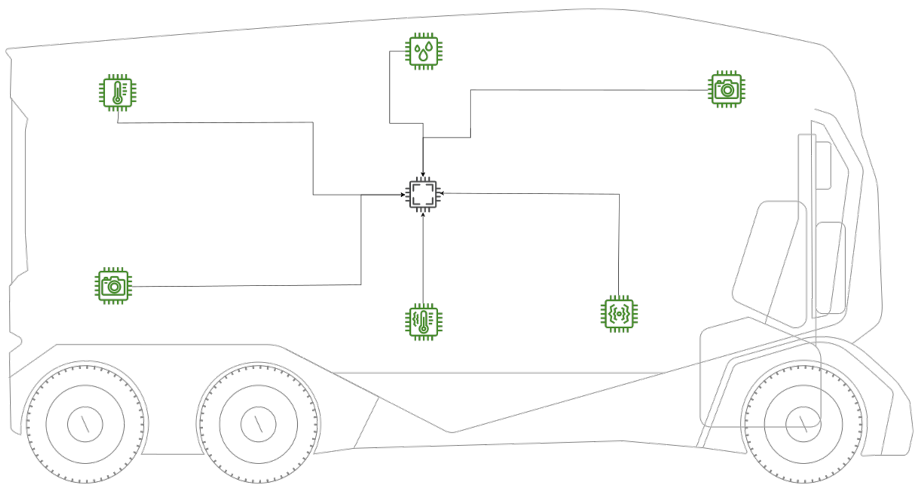

Figure 2.1: The sensor domain of the network is comprised of the autonomous drive sensors and a high performance computer connected with automotive Ethernet in a star topology.

2.2 Protocols

In this section, the different security protocols that have been assessed in this thesis are presented more thoroughly.

2.2.1 IPsec

IPsec is part of the IPv4 suite but was designed to be usable with the future IPv6. It supplies users with the capability to secure communication across LAN, WAN, and the wider Internet. It is commonly used to set up VPNs, for instance, to let employees work remotely over a secure link or establish extranet and intranet connectivity between organizations. The principal feature that allows IPsec to offer this functionality is that IPsec can encrypt and authenticate *all* traffic that is sent over the network layer.

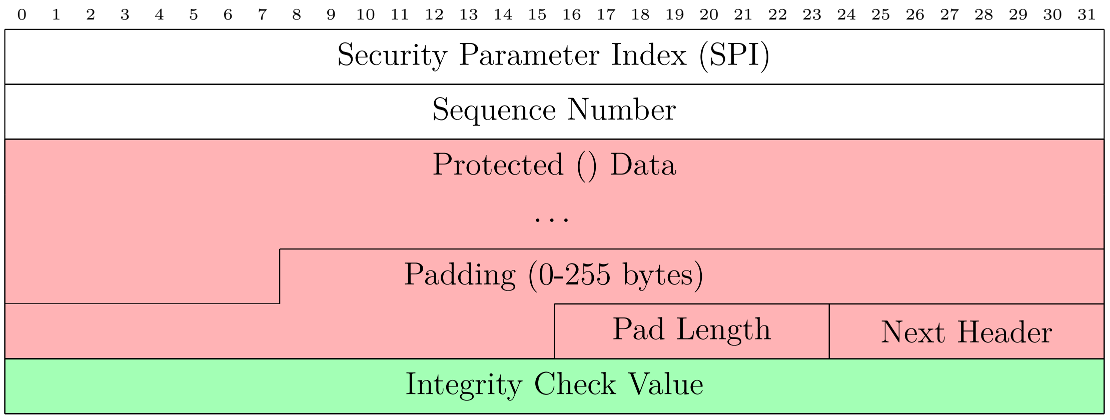

Figure 2.2: Top-level format of an ESP Packet.

IPsec operates in two distinct modes, Transport mode and Tunnel mode. In transport mode, only the packet's payload is encrypted and authenticated, leaving the IP header as it were. Since only the payload of the IP packet is encrypted, transport mode is generally used for end-to-end communication, for instance, between two workstations or a client and a server.

In contrast with transport mode, which only encrypts and authenticates the payload, tunnel mode encapsulates the entire IP packet in a new IP packet, protecting both the payload and the IP headers. Furthermore, the origin and destination headers can be changed from the original packet by creating a new packet. Tunnel mode is generally used when both ends of the connection are running IPsec, such as two routers running IPsec. This setup lets users on two separate unsecured networks communicate securely over a secured channel [19, pp. 666–667].

IPsec uses Encapsulating Security Payload (ESP) to provide confidentiality and integrity to its packets. It can also use Authentication Headers (AH) to provide authentication, but since ESP does that too, this extension's use has been deprecated. The format of an ESP packet can be seen in Figure 2.2. The Security Parameter Index identifies the Security Association, a logical connection between the sender and receiver that affords security services. The Sequence number is a counter that provides replay protection. The padding is an optional field dependent on the encryption algorithm used. Some algorithms require that the data it encrypts is a multiple of some block; this field can then be used to fulfill that requirement. The pad length indicates how many bytes the payload has been padded in width. The Next header stores the original header of the upper layer protocol since it gets replaced with ESP. Lastly, the Integrity Check Value is used to check if the payload has been altered. The length of this field depends on the authentication algorithm used [20, pp. 673–680].

2.2.2 MACsec

MACsec is a point-to-point network that works over a wired Ethernet link. It was standardized in 2006 as IEEE 802.1AE [21], but it wasn't until 2016 that it was incorporated into the Linux kernel with the introduction of version 4.6. MACsec works by securing Ethernet packets. It can secure a network from most attacks, including Man-In-The-middle, DoS, passive wiretapping, and playback attacks. It can also secure almost all types of traffic on an Ethernet link, including frames from the Link Aggregation Control Protocol (LACP), Address Resolution Protocol (ARP), Link Layer Discovery Protocol (LLDP), and Dynamic Host Configuration Protocol (DHCP).

A MACsec frame can be seen in Figure 2.3. It starts with the standard Ethernet header containing, among other things, the source and destination MAC addresses and an optional 802.1Q tag for VLANs. Next is the MACsec SecTAG containing the packet number and information to help the receiver identify the correct decryption key. Following the SecTAG is the payload, which can be encrypted. Lastly, the Integrity Check Value (ICV) is calculated with GCM-AES-128. The ICV is what guarantees that the packet was indeed created by a node in possession of the security key and that the packets have not been altered on the way [22, Sec. 9].

Except for the security of MACsec, the most significant benefit is that it is designed to be implemented in an application-specific integrated circuit (ASIC) on the PHY or switch card. This lets MACsec operate at line rate. This is crucial for networks where a lot of data is generated, such as an IVN. Unfortunately, due to it not being widely adopted yet, ASIC implementations of MACsec can be expensive [23, 24].

2.3 Key Agreements

This section briefly explains MACsec's and IPsec's companion key agreements.

2.3.1 Internet Key Exchange

For the generation and distribution of secrets keys IPsec utilizes IKE, or more specifically IKEv2 [25, 26]. The IKE key determination is based on the Diffie-Hellman Key exchange [27], with some additions for increased security. These additions have been made to mitigate some of the shortcomings of the original Diffie-Hellman implementation, such as its susceptibility to Man-In-The-Middle attacks, Clogging attacks, and Replay attacks.

In IKEv2 all messages are exchanged in pairs. The first two pairs are known as the

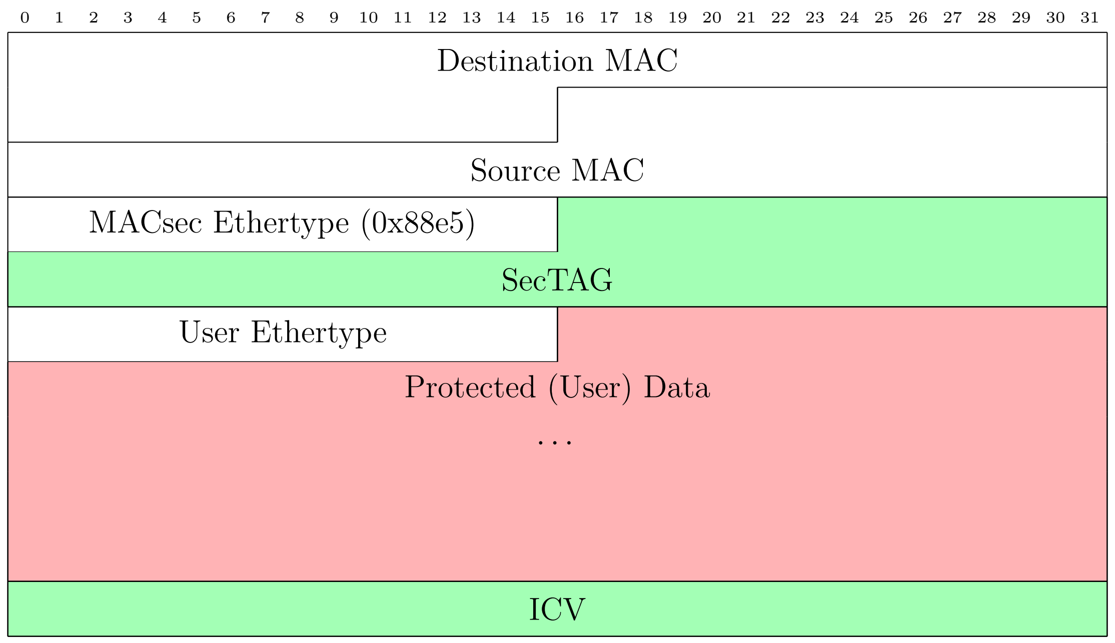

Figure 2.3: The layout of a MACsec frame.

initial exchange and involve exchanging information about which security algorithm to use, nonces, Diffie-Hellman values, and other security values. When all of this information has been exchanged, the hosts will have set up a special type of Security Association (SA) called an IKE SA. This SA is used for authenticating and encrypting the rest of the messages sent during the IKE. In the second exchange, the two parties authenticate one another and set up the first IPsec SA, thus protecting all traffic sent between the hosts [20, pp. 687–688].

2.3.2 MACsec Key Agreement

The MACsec Key Agreement (MKA) is an extension of the Extensible Authentication Protocol (EAP) [28]. The two participants first authenticate via EAP, thus becoming a secure connectivity association (CA) member. The MKA is then used to establish secret keys between the members, which MACsec uses to secure the traffic sent between them.

As defined in RFC 3748, EAP is a framework used to establish network access and authentication[29]. It comes with a set of protocol messages that can be used between a client and an authentication server. The term extensible comes from the fact that EAP supports many authentication protocols. The one that MKA uses is EAP Transport Layer Security (EAP-TLS), defined in RFC 5216[30]. EAP-TLS uses the handshake protocol that TLS uses. The client and server can then authenticate one another by using digital certificates.

The authentication will result in the members having a shared Connectivity Association Key (CAK) and Connection Association Key Name (CKN). These keys can then set up Secure Channels (SC) between the members. These Secure Channels comprise a series of short sessions known as Secure Associations (SA), each SA having its own Secure Association Key (SAK). These keys are then used to secure the traffic between the MACsec devices. When the SAK is about to expire, MKA swaps out the SA for a new one, thus refreshing the key [28, Sec. 9].

2.4 Cryptography

The different protocols and key agreements, presented in Section 2.2 and 2.3 respectively, utilize several different cryptography algorithms. This section gives an overview of the most prevalent cryptographic algorithms.

2.4.1 Advanced Encryption Standard

The Advanced Encryption Standard (AES) is a standard selected by the National Institute of Standards and Technology to replace the Data Encryption Standard. The actual mechanics of the algorithm was chosen through a competition held by NIST, and the final algorithm is a variant of Rijndael[31]. AES is a symmetric-key algorithm, thus requiring the same key for encryption and decryption. AES comes in three different flavors depending on the length of the key used. These are AES-128, AES-192, and AES-256, each used with the respective key size.

AES is a block cipher, meaning that it can only encrypt or decrypt data on a specific block size — in this case, 128 bits. Unless a message is precisely 128 bits long, there needs to be some function to split up a message and encrypt it in parts. The function for doing this while still guaranteeing message security is called modes of operation. The protocols in this thesis that use AES utilize a mode of operation called Galois Counter Mode. However, to explain how Galois/Counter Mode works, some knowledge about how Counter Mode works is needed [20].

Counter Mode (CTR) Counter mode entails using a counter in the encryption process to essentially convert the encryption algorithm into a stream cipher. The counter is randomly initiated and then incremented by one for each message, thereby making it unique for each message. Then, instead of using the counter as a key to encrypt the block itself, the counter is encrypted. This will create what appears to be a random string of 128 bits. These encrypted blocks will then be XORed with the blocks we want to encrypt.

This technique is good because it ensures that every block is unique from any other bock, even if the two blocks contain the same plaintext. An attacker can not learn anything about the data by simply observing the encrypted traffic. It is also realizable, making it faster than some simpler modes, such as Cipher Block Chaining or Electronic Codebook. However, it does have some weaknesses. For instance, it does not protect the integrity of the message [32].

Galois/Counter Mode (GCM) Galois/Counter Mode attempts to solve the shortcomings of CTR by appending a message authentication code (MAC) to every message. This lets the receiver check that the message has not been altered on the way. There also exists another version of GCM called Galois Message Authentication Code (GMAC) that creates the same MAC but without confidentiality [33].

2.5 Performance and Metrics

This section gives a brief overview of the three main metrics used to evaluate the security protocols.

2.5.1 Latency

Latency refers to the time it takes for a message to travel between two different hosts on the same network. It is commonly measured in one of two ways. The first way is a one-way measurement, measuring the time for a message to travel from one host to another. The other way latency is measured is by measuring the Round Trip Time (RTT). The RTT is the time between a message being sent and the time it takes to get a reply. Since this measures the time it takes for a message to travel both ways, the RTT is then divided by two to get the one-way latency. The function looks as follows

$$\frac{RTT}{2} = Latency.$$

While the one-way latency theoretically is more accurate, it requires the two clocks hosts that the test is performed on to have exactly synced clocks. This is very difficult to achieve. The most common way of measuring latency is with RTT, which is also how latency will be measured in this thesis.

Latency plays a vital role in an in-vehicle network. The autonomous drive models need to have close to real-time data to create routes for the vehicle accurately. If the latency increases too much, the data could be too old to be usable in the autonomous drive models. This would make the models discard the data or create faulty paths based on outdated data.

2.5.2 CPU Usage

The CPU usage refers to the amount of work that the CPU has to perform and is measured as an average over a certain amount of time. This thesis will measure CPU usage by the amount of time the CPU is actively doing work. For instance, if a single process is running and the CPU usage is measured at 10%, the CPU is spending 10% of its time actively executing that single process. That also means that the CPU is idle 90% of the time, and that time can be used to execute other processes.

The CPU usage of the security protocols is a highly relevant statistic, as the time that the CPU spends executing processes related to them is time that can not be used to execute processes related to other operationally critical processes. If the security protocols take up to much of the CPU's resources it could have an effect on the autonomous drive functions, which would be detrimental.

2.5.3 Throughput

Throughput is a measure of how much data can be processed in a given amount of time. Network throughput is the amount of data that can be successfully moved from one host to another within a given amount of time, usually measured in megabits per second or gigabits per second.

2.6 Zero Trust

A zero trust architecture is a framework that mandates all users to be authenticated, authorized, and continuously validated to get access to resources. This stands in stark contrast with traditional perimeter security, where strong perimeter firewalls create a sort of walled garden. Any entity that is withing this walled garden is seen as trusted, since it is assumed that no malicious actor can pass the secure perimeter. While this approach used to be somewhat secure, modern networks are generally no longer defined to a certain geographical location. The advent of cloud computing, remote work, and large spread-out businesses have scattered the resources. Due to this, more and more organizations have started to adopt a zero trust architecture. In fact, in January of 2022, the White House released a memo requiring all United States Federal Agencies to adopt a zero trust architecture [34].

The memo that was sent out by the White House contains a quote from the Department of Defense Zero Trust Reference Architecture:

> The foundational tenet of the Zero Trust Model is that no actor, system,

> network, or service operating outside or within the security perimeter is trusted. Instead, we must verify anything and everything attempting to establish access. It is a dramatic paradigm shift in philosophy of how we secure our infrastructure, networks, and data, from verify once at the perimeter to continual verification of each user, device, application, and transaction [35].

This notion of not trusting any resource on the network will be the definition of zero trust that is used throughout this work.

3 Threat Model

One way to reason about the security and possible threat to a system is by using threat modeling methods. A threat model is a structured approach to identifying possible threats within a system. These threats can then be prioritized, and the value of mitigating them can then be determined. Threat modeling is a good way to incorporate security early in a design phase and is used diligently in many industries. While threat modeling has not generally been used to model vehicles as they have been seen as an enclosed system, research has been done to determine good threat models for connected vehicles [36].

While no structure threat modeling has been done for this work, the move to a zero trust architecture is meant to secure the vehicle for one certain scenario — a malicious actor getting access to the internal network. This could be done either by inserting a malicious device into the network, connecting it to an automotive Ethernet switch, or replacing a sensor. This is not an unlikely scenario. As the vehicles will drive autonomously, they will be left unattended for extended periods of time. This will give the malicious actor plenty of opportunities to access the vehicle.

If the actor manages to breach the internal network, three main attacks have been determined as the most valuable to mitigate: message spoofing, man-in-the-middle, and replay attacks. These three attacks have been prioritized due to a combination of them being easy to perform and having a large impact on the vehicle's usability. Sections 3.1, 3.2, and 3.3 will give a deeper explanation of how these attacks function and how they are mitigated.

3.1 Message Spoofing

Message spoofing is an umbrella term for a couple of different attacks, such as IP spoofing, MAC spoofing and ARP spoofing. What they all attempt to do is impersonate another device on the network. For example, a basic security measure is to set up a firewall to protect devices from getting messages from unknown sources.

Among other things, these firewalls can set up a whitelist, where only a select number of IP addresses are allowed to pass through the firewall. Since an IP address is associated with a device, the optimistic assumption is that any traffic coming from this IP address is coming from a known device. However, the IP address that is marked as the sender can be faked by a malicious party, thus letting it send arbitrary messages to the device. MAC spoofing and ARP spoofing work in the same way, in that they try to impersonate another device, but for different protocols and with different results.

To protect against these types of attacks, IPsec and MACsec both employ integrity check values (ICV). An integrity check value is a string attached to the end of a packet and calculated from the rest of the packet's data. When the packet arrives at its destination, the ICV is recalculated and checked against the value attached to the packet. What differentiates this process from CRC is that instead of using a common mathematical function to calculate the string, IPsec and MACsec utilize cryptographic hash functions to calculate the string. More specifically, IPsec utilizes sha256 (HMAC version) while MACsec utilizes GCM-AES-128. These functions take in the message and a secret key shared between the two communicating devices and output a unique hash. If any part of the message changed, the hash would come out differently. And since only the two communicating parties know the secret key, the ICV can't be recalculated by a third party.

While providing good security, the calculation will infer some added overhead, both in terms of data latency and CPU usage, but also with extra data. The latency and CPU usage come from the added operations that must be performed on both ends of the communication line. The sending party has to produce the ICV and attach it to the message, and the receiving party has to recompute it and compare the two to verify the message. How much of an increase in latency and CPU usage comes down to the hardware that is being used. Some CPUs have built-in operations for certain cryptographic functions, such as most Intel chips with their AES-NI operations. If a device has access to cryptographic hardware that lets them offload the operations, it will significantly reduce CPU usage and latency. In the case of MACsec, it has been designed to be implemented into hardware, thus removing the added CPU usage and latency altogether.

However, the added data will have an effect regardless of the underlying hardware. The ICV that MACsec adds on is 16 bytes. The standard maximum transmission unit in Ethernet is 1500 bytes - some networks support so-called jumbo frames with a size up to 9000 bytes, but they will not be part of this discussion. Therefore, the 16 bytes of the ICV only add about a 1% overhead. However, an Ethernet frame can be as small as 64 bytes. In this scenario, the ICV will add a 25% overhead to every frame sent. The ICV produced by IPsec's SHA256 hash is also 16 bytes (truncated down from 32), but a maximum packet size of an IP packet is 65,535 bytes. In this case, the addition of 16 extra bytes is inconsequential. On the other side of the spectrum, adding 16 extra bytes to the smallest packet size of 21 bytes will add a 76% overhead to the overall message.

3.2 Man-in-the-Middle

A Man-in-the-Middle (MITM) attack is a cyberattack where a malicious third party inserts itself between two communication parties, relaying and possibly altering the data transmission between them. If done successfully, the two communicating parties will be unaware of the intermediate third party - believing that they are communicating with each other. A common version of the MITM attack is active eavesdropping. In this scenario, the malicious third party makes independent connections to the two parties while masquerading as the opposite communicator. This will allow the third party to transparently intercept all traffic, even if it's encrypted, and then relay it to the unsuspected participants.

Protocols can use encryption or hash functions to protect against snooping or message alteration. However, if an attacker is able to insert itself during the initial establishment of the secure communication channel this will not help. Therefore, the key exchange also has to protect against this type of attack. Several techniques have been developed for this purpose, such as public-key authentication, pre-shared-key authentication, and the extensible authentication protocol. IKEv2, used by IPsec, supports all of these authentication methods and can even layer them on top of each other with the RFC 4739 extension. While MACsec also supports pre-shared keys, since the MKA is an extension of IEE 802.1X, utilizing EAP is much more common. For a description of EAP and 802.1x, see 2.3.2.

These solutions do not add any overhead to the communication between the two parties, but they come with their downsides. The utilization of pre-shared keys, while simple, is not scalable as every connection has to be manually configured. Public Key Authentication and EAP do not suffer from scalability issues, but they add some complexity to the network. Especially EAP with its three different components, all of which have to be correctly implemented and configured. If an exploit in one of the parts was discovered, it could jeopardize the entire network's security.

3.3 Replay Attacks

A replay attack is an attack in which valid traffic is maliciously retransmitted or delayed. Since the traffic being retransmitted originated from a trusted source, the included ICV will be correct, thus making the receiver trust the data. One instance where replay attacks have been used successfully is the remote keyless system included in many modern cars. An attacker would record the radio waves transmitted from the remote key to unlock teh vehicle, which could then be used to open the vehicle later.

Both IPsec and MACsec utilize packet numbering to counter replay attacks. Both parties of a communication association have two counters each, one for incoming packets and one for outgoing packets. When a packet is transmitted, the corresponding counter is incremented, and the number gets attached to the outgoing message. When the other party receives the packet, the corresponding counter is incremented. While the message could be checked against this exact counter, both protocols have the option to make use of a “sliding window" instead. This window contains a lower bound, in which a message is too old, and all message numbers that have previously been verified. By utilizing a sliding window, packets are allowed to come out of order. When a message gets verified, the sliding window gets updated and shifted one step to the right.

Packet numbering adds little overhead to the actual packet, only consisting of 4 bytes for both IPsec and MACsec. However, the refreshing of the secure keys, needed to keep the numbering secure, does add some overhead. The packet number system allows 2^{32} packets to be secured with a single key. When all those numbers have been spent for the current key, it has to be exchanged to prevent the reuse of sequence numbers. This key exchange will add a small overhead, but not any that will affect the system. What it does do is add another layer of complexity that could introduce bugs.

4 Literature Overview

The discussion around vehicle security has been ongoing for a long time, both for automotive Ethernet and legacy bus protocols. Research has been done in both fields, and more holistic attempts have been done to map the entire attack surface of a vehicle. This section contains the relevant literature for this thesis and what differentiates the work in this thesis from the previous work.

There have been many papers discussing the attack surface of vehicles [37, 38, 39]. These cover everything from Bluetooth hacking, Wi-Fi hacking, and keyless entry systems. Most of these papers are focused on consumer vehicles, which have attack vectors that are out of scope for this thesis. Excluding the previously mentioned attacks, much focus has been placed on the On-Board Diagnostics (OBD-II) port. The OBD-II port is a mandatory port that lets independent mechanics read and write to the internal network. The purpose of the port is for mechanics to take diagnostics of the car, but since it has full access to the internal network, it gives a potential attacker full access to all the internal ECUs.

While the more general security overview papers have at least a discussion about Ethernet as a network medium, they are usually short, only focusing on some of the benefits but rarely specifics. There is, however, research that is aimed explicitly at Ethernet in the automotive industry and to discuss its security flaws [40, 41, 42, 43]. While not suggesting any particular solution, they all emphasize the importance of message integrity to secure the system against spoofing and tampering. Confidentiality is also mentioned as a good security practice but not in favor of message integrity.

Lang tried, to some degree, to solve some of the same issues that this thesis is attempting to solve — message integrity in time-critical systems [44]. The approach this paper takes is to test different types of HMAC algorithms and then see if the added latency is within the requirements of the thesis. However, all the tests are done in simulations, and the authentication is implemented on the application level of the OSI model. Furthermore, it also measures data from the sensors to the breaks, a distance different from the scope of this work. While it does come to some satisfactory conclusions, with HMAC adding only about 2% more latency, this thesis will focus more on creating a holistic solution than measuring latency for individual algorithms.

There has also been research that has focused on a more general approach for securing Ethernet in automotive solutions. Lastinec and Hudec looked at TLS, DTLS, and IPsec as possible solutions for securing the stream [45]. The paper found that DTLS and TLS were not suitable for in-vehicle networks due to the immense latency increase, while the best result came from UDP over an IPsec secured channel. However, this paper evaluated CAN traffic that used Ethernet as a backbone. The amount of traffic sent over the channel is therefore not comparable to a sensor network sending hundreds of megabits over the network every second.

MACsec has also been mentioned as a possible solution for securing the IVN. Lauser *et al.* mentioned them as a potential future solution and recommended an analysis of the protocol as future work [46] . H. Y. Lee and D. H Lee also suggests MACsec as a possible solution for network security, but refrains from it due to the hop-by-hop nature of the protocol [47]. This issue has been discussed in a paper by Choi *et al.*, who solved the issue by extending MACsec over a Software Defined Network [48]. However, LAN bridging will not be an issue in this thesis as all traffic is limited to a single LAN. In fact, this limitation can be an advantage since it will add an extra barrier to bridging VLANs, thus strengthening segmentation between the sensors.

As explained in Section 2.2.2, one of the main benefits of MACsec is that it can be implemented completely in hardware to secure the network without any performance loss. Carnevale *et al.* showcase a hardware implementation of MACsec, specially constructed for INVs [49]. They managed to obtain a throughput of 3.9 Gb/s on an FPGA implementation. Han *et al.* also developed a hardware implementation of MACsec, managing to achieve line-rate throughput (1 Gb/s) on an ASIC card [50]. However, there have also been attempts to optimize the Linux kernel's software implementation. Lackorzynski *et al.* tested the effect of fragmentation and different ciphers to increase the performance of MACsec only run on the CPU [51]. They found that the use of ChaCha20 [52] significantly increased the throughput on devices without AES acceleration compared to the standard AES-GCM cipher.

5
Methods

This chapter presents the methods and metrics used to fulfill the research questions posed in Section 1.2. Section 5.1 presents the performance requirements that the protocols have to conform to in order to be used in an automotive setting. Section 5.2 presents the tools and tests performed to analyze the network configurations. Section 5.3 presents the implementation of the Bump-in-the-Wire device and key exchange. Lastly, Section 5.4 details the three different test benches, their use, and what tests were performed on each one.

5.1 Performance Requirements

To assess if a certain protocol is performant enough to be viable in an IVN, some performance requirements have to be determined.

5.1.1 CPU Usage

To not impede the performance of the vehicle's autonomous driving, some requirements have to be set for the amount of CPU that the protocol is allowed to use. This limit has been set to an increase of 5% compared to the baseline CPU usage. This amount will not deter from the vehicle's overall functionality while still not being so small that it can not make an impact. Security has to be considered a priority, and the allotted CPU load should reflect that.

5.1.2 Throughput

Due to the large amount of data generated every second, the network's throughput is an important aspect. If the protocol were to limit the amount of data that can be transported on the network, that could eventually start restricting the types and amount of sensors that the vehicle can carry. The introduction of a security protocol could affect throughput in several ways. For instance, it could add overhead to the data sent, and increase latency, thus reducing the amount of data that can be transferred in a certain interval. Therefore, the security protocol should not decrease the throughput by more than 5%, compared to the throughput of a network without the protocol.

5.1.3 Latency

To be able to effectively use the data that is being transmitted to the main computer, it has to be up to date. Old data can cause the vehicle to behave in ways it should not. This means that all the data that travels in the system should be as close to real-time as possible, and any data that is just a few milliseconds old will be discarded. After consulting with one of the sensor teams at Einride, a threshold of two milliseconds was decided on. If the protocol keeps the latency below 2 ms for 99.9% of all packets sent, it could be a viable option to implement in an autonomous vehicle.

The reason that 99.9% of packets should clear the 2ms threshold, instead of having the median pass a lower threshold, is to check if any of the protocols could introduce a long-tail latencies. A long-tail latency is when high percentiles of latencies, 90th percentile and above, start having latencies significantly higher than the average.

While this will not affect a majority of packets, an in-vehicle network has such a high load that it can still have significant effects on the system. For instance, if 99% of all messages clear the 2ms threshold, every 1 in 100 packets will not pass it and be too old to be used. This could also be framed as a 1% packet loss, which is more than enough to impact the autonomous drive system.

5.2 Testing

The testing aims to measure the performance metrics described in Section 5.1 on different protocols and network configurations. Three different types of tests have been created to measure each of the three types of metrics. All of the tests have then been run with three sensors concurrently performing the tests on the consumer, making the system more closely resemble a production environment.

The three tools used when performing the tests are Sockperf[1], System Activity Report (Sar)[2], and iPerf3[3]. Sockperf is maintained by Nvidia and was created to

[1]`https://github.com/Mellanox/sockperf`
[2]`https://github.com/sysstat/sysstat`
[3]`https://github.com/esnet/iperf`

measure latency throughout high-performance systems. It supports multiple modes for measuring latency and throughput and also has an option to replay recorded data. On the other hand, Sar is a tool for measuring system activity on the device. It is part of the sysstat suite of tools. Finally, iPerf3 is used for measuring throughput. It is an open-source software written in C and maintained by ESnet, a part of the United States Department of Energy. It is widely used in academia [53, 54, 55, 56]. The tool works by setting up a server and client on two different machines, which will perform the throughput test. The user can then set fine-grained configurations on how the test should be performed.

5.2.1 Latency Test

When measuring latency, three different types of tests are performed with Sockperf. The first test is described as “Under Load” by the Sockperf manual. This test is meant to simulate a system under load by consciously sending packets to the server-side. On every 100th message, a reply will be sent to the client-side with information about the latency. Since the message is sent back to the client so rarely, it should not affect the one-way stream, thus creating a more realistic simulation. As the latency should reflect the production environment, the test has been modeled after recorded sensor data. The packets that are sent have a size of 24938 bytes, and they are sent at a rate of 950 messages per second. This message size is the same as a Lidar packet. This simulation is run until 200 000 measurements have been made.

The second measurement technique, called Playback, works more or less the same way as the first one, but recorded data is used as input instead of just setting parameters for the test. The program then sends packets at the same time intervals as the recorded file, leading to a very accurate simulation. The last test, known as Playback, is a more traditional latency test, where every message sent will have a response. Again a packet size of 24938 bytes is used to match a Lidar packet, but the message per the second definition has been removed.

5.2.2 CPU Test

The CPU test measures the resources used by the entire system under the highest load that three sensors can generate. To achieve this, Sockperf’s throughput mode is utilized with a packet size of 1440 bytes. This packet size has been chosen to avoid any fragmenting. The Maximum Transmission Unit is set to the default 1500 bytes. But since MACsec adds 32 bytes, the UDP packet adds 8 bytes, and the IP packets adds 20 bytes the total amount of data that can be transferred without fragmentation is 1440 bytes.

The test consists of the sensor performing the throughput test for 50 seconds and then pausing for 20 seconds. The pause is in place to let the system recover and clear any buffer or cache that might affect the upcoming test. The test is then be repeated a total of 100 times. A Sar instance records the CPU usage during the entire testing duration by taking snapshots every second. When the test has finished, the recorded data can be analyzed to get the median CPU usage by any given protocol.

5.2.3 Throughput Test

The throughput test is performed on three devices simultaneously to account for any possible effect that the added CPU load might have. Three instances of the iPerf3 server is set up on the main computer and mapped to three different network interfaces, one for each sensor. The sensors are then start the test in unison. Instead of the default TCP, the tests are performed with a stream of UDP packets. As the Ethernet ports are specified to be able to handle 1 Gbit/s, the bandwidth is set to just that.

5.2.4 Security Test

While both the IPsec and MACsec standard guarantees message integrity [22, Sec. 6.9][57], the implementation's conformity to these guarantees should still be tested. Therefore, some security tests are performed to check the security of the network configurations. The tests that are performed cover the most general attacks discussed in Section 3: message spoofing and replay attacks. While it would be good to test the network against man-in-the-middle attacks, it has been left out due to its complexity compared to the other two attacks. It is instead left for future work.

The tests are performed with a single packet that has been modified for the specific attack in question. The tests performed on IPsec use a modified IP packet, and for MACsec, it is a modified Ethernet packet. The first attack sends a single UPD packet to the protected address. No modification have been done to this packet. The IP packet contains the malicious device's IP address, and the Ethernet packet contains the malicious device's MAC address. This test mostly confirms that the security protocols have been correctly configured. The second test use packets with the source headers changed. It is still a normal IP or Ethernet packet but with correct IP or MAC source addresses. While these first two attacks should be protected by the security protocols, they could also be averted by simply configuring a firewall to block all traffic that is not of the ESP or MACsec type.

The third attack is a replay attack. In this attack, the packet that is sent is a legitimate packet that has been captured with tcpdump[4] on the receiving end of a

[4]`https://github.com/the-tcpdump-group/tcpdump`

communication channel. It is therefore completely correct, and the ICV will check out. However, the packet number will not be up to date, and the packet should consequently be dropped. Two versions of this attack are be performed. When the replay window is set to one, ensure that the packet is not within that window, and one where it is set to 1024, to make sure that the packet is within the allowed replay window. This ensures that arrived packet numbers are registered correctly so that they can not be replayed within the replay window.

The final test is performed with the same packet as before, but with the transmitted data slightly changed. This should make the ICV check fail, thus making the protocol drop the package. To ensure that the ICV check catches the error, replay protection is turned off. These four tests will cover the fundamental attacks that could be performed on the internal network. The tests are far from exhaustive and an investigation into the implementation of the two protocols might be interesting. However, this is left up to future work.

5.3 Implementation

This section goes through how the Bump-in-the-Wire and the key exchange were implemented. Due to time constraints, they were only implemented and tested for MACsec. However, it should be noted that the general idea for the key exchange is also applicable for IPsec.

5.3.1 Bump-In-The-Wire

The implementation of MACsec in the Linux kernel works by creating a new network interface with the type of MACsec, macsec0. This interface is then connected to a real network device of type Ethernet, eth0. Any configurations on what security should be enabled are then configured on the macsec0 interface. This has to be done on both communicating devices. After a successful MACsec setup, secure traffic can be exchanged between the two macsec0 interfaces. The traffic will travel through the eth0 interfaces with the additional MACsec header. The MACsec header count towards the eth0 MTU, so the macsec0 MTU has to be reduced by 32 bytes.

Another type of network interface present in Linux is the Bridge interface. This interface works like a network switch, transparently forwarding network packets on the Link Layer based on their MAC addresses. Unlike a hardware switch where a user connects networks by ethernet cables, a software bridge connects networks by setting it as a master to network interfaces. Since MACsec is implemented by creating a network interface, macsec0, it can be connected to a network bridge.

If a second Ethernet interface, eth1, is connected to the bridge, it will connect the two networks. This step will let the eth1 interface forward packets through the bridge and then to the macsec0 interface. Here the MACsec headers will be added, and the packet can then be securely transmitted to the corresponding MACsec device. However, since the MACsec header will add 32 bytes to the message, the eth1 and br0 interfaces have to have their MTUs lowered to 1468 to account for that and any device that wants to communicate with the eth1 interface.

This setup, which is depicted in Figure 5.1, lets a device that is connected to the eth1 interface transparently communicate with the MACsec enabled device and have all its data secured between that device and the Bump-in-the-Wire.

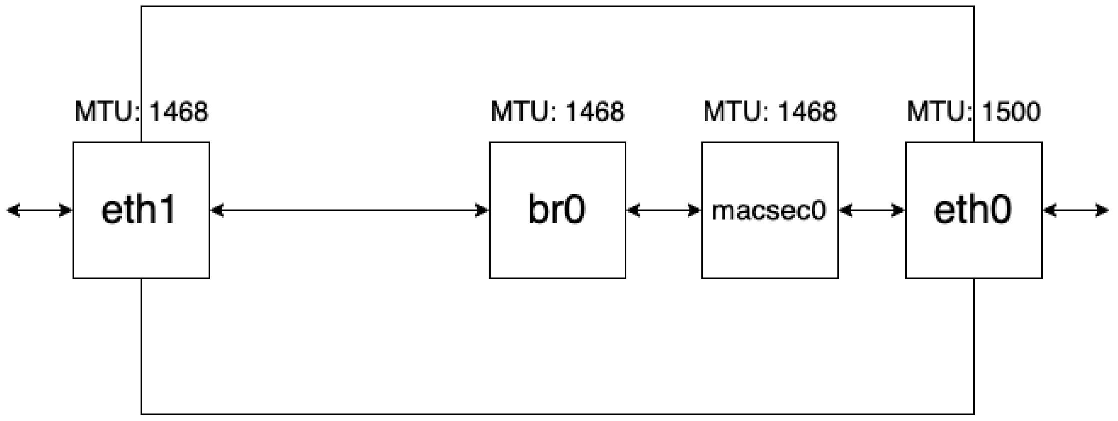

Figure 5.1: A depiction of the Bump-in-the-Wire implementation.

5.3.2 Key Exchange

The key exchange that was configured relies on the implementation of the MACsec Key Agreement, explained in Section 2.3.2. As discussed in that section, 802.1X, which MKA is an extension, relies on three parts: a supplicant, an authenticator, and an authentication server. In this scenario, the supplicant is the intermediate device or sensor, which will attempt to attach to the network in the main computer, acting as both the authenticator and authentication server.

For the supplicant part of the exchange, wpa_supplicant was used. This is a free software implementation of IEEE 802.11i supplicant included in several Linux distributions by default. Furthermore, it can be compiled to include a MACsec driver that can negotiate master session keys via MKA, either using pre-shared keys or X.509 certificates. Moreover, it also handles any renegotiations after the key has expired.

The authenticator used is hostapd, an open-source tool that comes in the same bundle as wpa_suppilcant. This program comes with the same driver as wpa_supplicant, making it possible to perform MKA with hostap as the authenticator. The RADIUS server that is used is called FreeRadius. Just like with wpa_supplicant and hostapd, it is also an open-source product.

A new Certificate Authority (CA) is created and used for signing the server and client certificates. The key for the CA is kept outside of the actual network. While it could be held in the main computer, thus distributing the certificates from one another, it causes more trouble than good. When a new certificate needs to be created, the vehicle's internals have to be accessed, which might not always be possible. It might also cause other security issues, as the connection to the vehicle has to be made safely. However, if the certificates are stored outside of the vehicle, they can be kept in a location with robust access control, making it more secure and convenient to create new certificates.

One server certificate is created for the RADIUS server and one each for every connected client. The certificates that are created are of the X.509 standard and are stored in Privacy Enhanced Mail (PEM) files, defined in RFC 1422 [58]. While testing, the public certificate and the private key will be stored in the same file, which is not advisable. In a production environment, these files should be split from one another and stored in a Trusted Platform Module (TPM). Another file format that might be interesting to store the files in is Public-Key Cryptographic Standard (suffixed by .p12), which, unlike PEM, is fully encrypted.

5.4 Test Bench

All the tests have been run on hardware to get more accurate results. The hardware consists of some less powerful Intel Next Unit of Computing (NUC) computers and a computer with similar characteristics to one of Einride's main autonomous drive computers. The NUCs are equipped with an Intel Core i3 processor and a single 2.5 Gigabit Ethernet port. The autonomous drive computer is equipped with an Intel i7-9700E CPU and six Gigabit Ethernet ports. In addition to the compute units, three TP-Link TL-SG105 Gigabit Ethernet switches are used when needed. All devices are running BalenaOS, an operating system created for IoT and optimized for running Docker containers [5]. The cables used are CAT-6 ethernet cables with RJ-45 adapters. While not automotive grade, they should be just as performant.

5.4.1 Testbench One

The initial testbench was used to get the baseline reading of the three network setups. The NUCs acted as the sensors, and the autonomous drive computer acted as the consumer. The tests were run concurrently on the sensors to make the system handle more data per second. During the throughput tests, the consumer had to

[5] `https://github.com/balena-os`

handle up to 3 GB/s of data. The first tests were run without any security protocol configured. The readings from these tests served as the benchmark with which all of the following measurements were compared to. This setup is representative of a production system, with multiple sensors sending concurrent data into multiple ports without any added overhead.

The second test was run with IPsec configured. Due to it being open-source, the Strongswan implementation of IPsec is used. Since this first test is only to measure the performance of the IPsec protocol, all the keys have been distributed manually and are not unique between the devices. Furthermore, the IPsec has been set to transport mode, and the encryption has been disabled. The configuration can be is seen in Figure 5.2.

The last test was run with MACsec configured on all devices. Like with IPsec, the keys were distributed manually and are not unique to the devices. Encryption was turned off, but packet numbering and the integrity check were turned on. The Linux implementation has an option to allow hardware offloading. Unfortunately, none of the devices used in the setup has the supported hardware, so this option was also turned off.

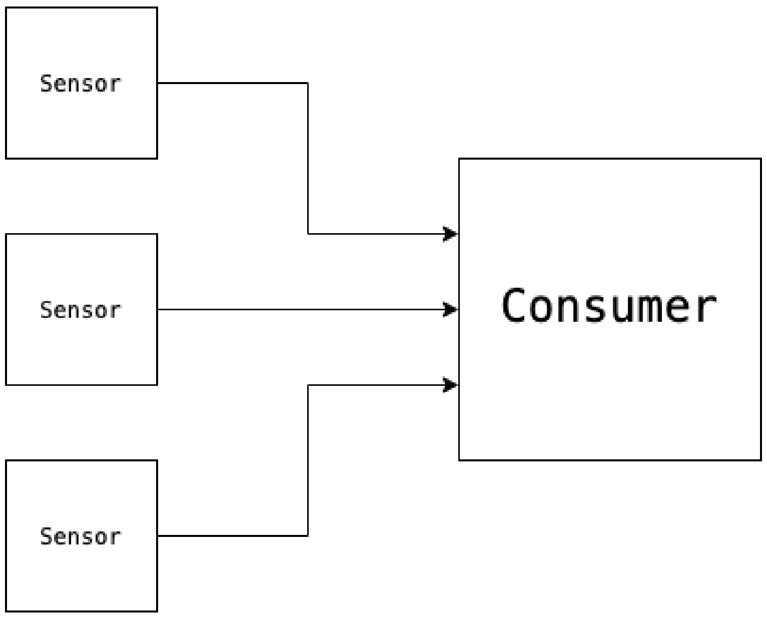

Figure 5.2: The first tesbench is used test the performance of the software implementations of the security protocols.

5.4.2 Testbench Two

The second setup tests how much impact the performance of intermediate devices, also called a Bump-in-the-Wire, have on the system. Even if a sensor does not support any security protocol, it should still be secured. The best way to retrofit a security protocol is with a Bump-in-the-Wire implementation. A Bump-in-the-Stack solution, where the software is implemented into the sensor itself, is not a scalable solution. The configuration would have to be customized to every single type of sensor and every iteration of the sensors themselves. The performance would also vary greatly between the devices, as some have specialized chipsets to perform one specific task. Therefore, a Bump-in-the-Wire solution is more scalable, as it can be attached to any sensor that transfers data using ethernet.

The configuration can be seen in Figure 5.3. It consisted of two NUCs and the main autonomous drive computer. One of the NUCs acted as a sensor without any security configurations, and the other NUC acted as the Bump-in-the-Wire. Lastly, since the NUC only has one ethernet port, a Gigabit Ethernet adapter was used to connect the sensor. While not optimal, it should still be as performant as a normal ethernet cable.

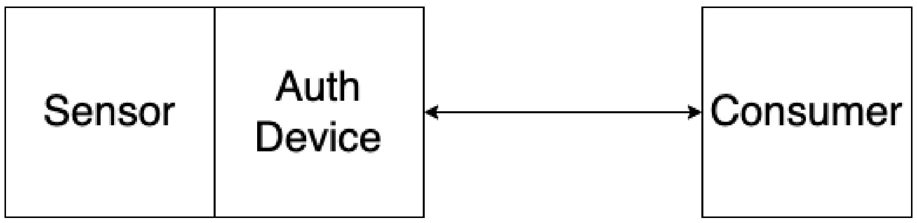

Figure 5.3: The second testbench used to test the performance effect of an intermediate device.

5.4.3 Testbench Three

The third testbench configuration, see Figure 5.4, mimicked the actual network setup that can be found in a production vehicle. In contrast to the first testbench, the devices were not connected directly to the consumer. The traffic was instead routed through a series of switches. The sensors were connected unevenly on the network, with some switches having more connected sensors than others and some having no connected sensors at all. Lastly, the switch that connected to the consumer did so with three Ethernet cables. While only one connection could have sufficed for the sensors to send messages to the consumer, this setup made it possible to spread out the traffic. Furthermore, it also opened up the possibility for some different tests on the network.

The reason for there being multiple switches instead of simply one is redundancy. While redundant switches might not be present in all in-vehicle networks, either due to space, cost, or some other factor, it is desired. Therefore, testing that the security setting does not compromise the viability of implementing multiple switches is essential. Thus, by having a redundant switch configuration, it can be verified that the normal flow of traffic is not altered when a security protocol is configured. Furthermore, a test was performed where randomly selected Ethernet cables were removed to simulate an outage. If the security protocols work as expected, the traffic should be rerouted when this happens.

This testbench was also used when performing the security tests described in section 5.2.4. The required traffic was collected during normal runtime and edited on a separate computer. This computer was then connected to one of the switches, and the tests were then performed. However, in addition to these penetration tests, some tests were also performed on the configuration of the network. The sensors were set up to communicate with one port each and should therefore not be able to communicate with any other port. For instance, if sensor A is configured to communicate securely with port A, and sensor B is set up to communicate with port B, sensor A should not be able to send messages to port B. Furthermore, the same test was also performed with Sensor C configured to the same port as sensor A.

Furthermore, the Bump-in-the-Wire configuration detailed in section 5.3.1 was also tested on this configuration. This was done to verify that it worked in a configuration containing devices that have the security protocol integrated by default. These devices should not be affected in any way, and no additional configuration should be needed. However, the sensor attached to the Bump-in-the-Wire needed to have its MTU reduced to account for the additional overhead.

Lastly, the key exchange described in section 5.3.2 was also tested on this network configuration. Separate certificates were created and distributed to the server and the sensors, and a central radius server was set up in the consumer. Two different configurations were then tested. The first configuration had the sensors connected to separate ports, thus creating three separate communication channels. The second had two sensors connected to the same port, therefore testing the implementation of group CAs in HostAP.

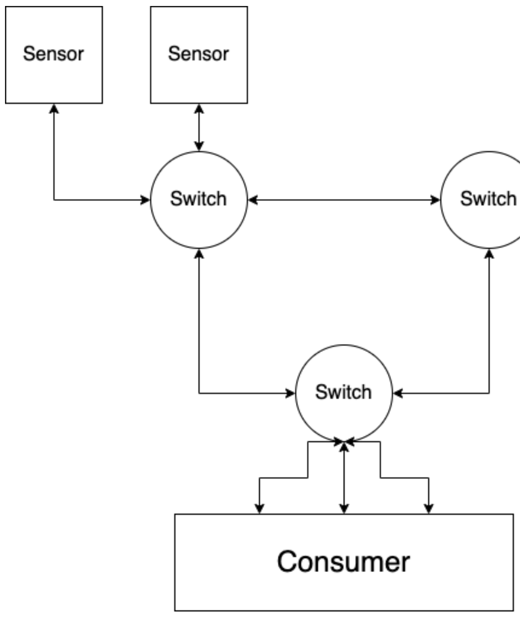

Figure 5.4: The third testbench used to simulate a production environment.

6
Results

This chapter details the results of the tests performed on the four network configurations described in Chapter 5. Section 6.5 also provides a comparison of the result in regards to the metrics defined in section 5.1. Section 6.6 goes through the results from the security tests.

Only the most relevant results are included in this chapter. For an exhaustive list of all the measurements taken during testing, see Appendix A.

6.1 Baseline

This section details the results from the tests done without any security configurations on the network. The results here will serve as a baseline and all the following tests will be compared to the results in this section. Note that performance tests were only performed on the first testbench and not with an intermediate device.

6.1.1 Latency

Figure 6.1 shows the distribution of the latency measurements that were taken during the Ping Pong, Playback, and Under Load tests. As shown in the figure, the network configuration without any security protocols shows no indication of a long-tail latency problem. This can also be confirmed by Table 6.1, with the highest reading being the 99.9th percentile of the Playback tests. This measurement is 62% higher than the median latency of the same test, which is 477 µs, which is not an unreasonable amount.

These tests have provided a baseline latency for the system of about 730 µs, giving the IPsec, MACsec, and Bump-in-the-Wire configurations around 1270 µs of available overhead.

Table 6.1: Maximum latency measured for the 90th, 99th, and 99.9th percentile with no security protocol enabled.

(µs)	Ping Pong	Playback	Under Load
90th	606	591	569
99th	713	670	622
99.9th	732	768	665

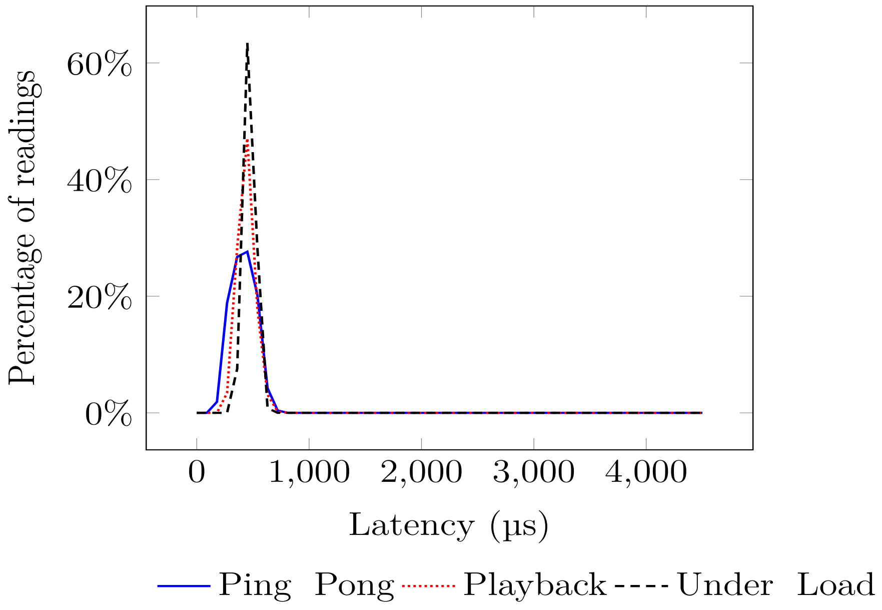

Figure 6.1: Latency distribution with no security protocol enabled.

6.1.2 Throughput

The throughput tests show a maximum throughput for the network without any security configurations at 956.1Mb/s, as seen in Table 6.2. This is close to the theoretical maximum throughput of 957Mb/s. The theoretical throughput is calculated by

$$\frac{payloadsize}{packet\ size} * bandwidth,$$

which in this case is

$$\frac{1472}{1538} \cdot 1000\ Mb/s = 957.1.$$

	Throughput
Baseline	956.1Mb/s

Table 6.2: Baseline throughput.

The packet size of 1472Mb comes from taking the standard MTU of 1500 bytes and subtracting the IPv4 header and UDP header, 20 bytes and 8 bytes, respectively. The Ethernet-specific overheads are not counted towards the MTU and will be added on top of it. First is the Ethernet header (18 bytes) and the CRC (4 bytes). Ethernet also has a preamble (8 bytes) and a minimum inter-packet gap (12 bytes), bringing the total package to 1538 bytes.

6.1.3 CPU Usage

Figure 6.2 shows the average CPU usage during a throughput test on the network without any security protocols enabled. The average CPU usage, excluding the first and last 10 seconds of the test, was 2.27%. The highest reading after averaging all the measurements was 2.57%. However, the highest reading from all the readings measured a CPU usage of 12.59%. One observation that can be made is that most of the CPU resources were allocated to kernel-level calls, including hardware and software interrupts.

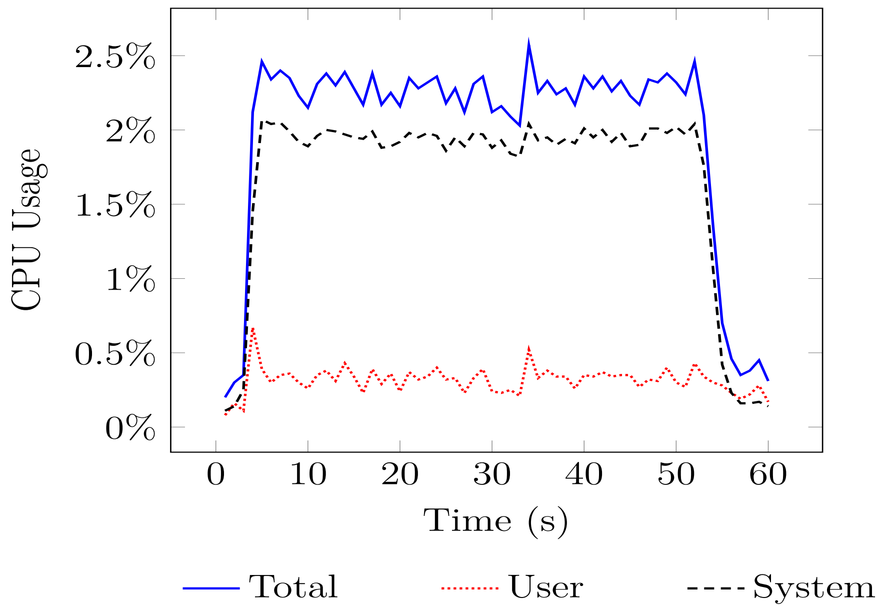

Figure 6.2: CPU usage without a security protocol.

6.2 IPsec

This section details the results from the tests done with IPsec configured on the network. The IPsec setup utilized pre-shared keys that were the same between all the nodes on the network.

6.2.1 Latency

Figure 6.3 shows the distribution of the latency measurements that were taken during the Ping Pong, Playback, and Under Load tests with IPsec configured on the network. By inspecting the graphs, it is not directly apparent if the measurements show a long-tail or not. However, if the long-tail exists, it will be in the highest percentiles of the Playback or Under Load tests, as they can be seen to have a slight curvature at the backend of their readings. Interestingly, the Ping Pong tests have a generally higher latency than the other two tests.

To confirm or deny the existence of a long-tail, Table 6.3 has to be taken into account. The table does prove the existence of a long-tail that is above the threshold in the Playback test, but not for the other two tests. This may have come about due to a smaller sample size, which might exaggerate the impact of an outlier. And since the other two tests do not show any indication of the existence of a long-tail, it is more likely that the long-tail does not exist. Another indication that this is the case is the significant increase in latency between the 99th and 99.9th percentile, especially compared to the median. The 99th percentile is 42% higher than the median latency of 955µs, while the 99.9th percentile increases 256%.

(µs)	Ping Pong	Playback	Under Load
90th	1383	1114	1212
99th	1479	1361	1468
99.9th	1677	3556	1621

Table 6.3: Maximum latency for the 90th, 99th, and 99.9th percentile with IPsec enabled.

6.2.2 Throughput

The tests show that the throughput for the system with IPsec enabled is 938.5 Mb/s, as seen in Table 6.4. IPsec communicates with ESP packets, which comes with some special header that causes a bit of overhead. This can be compared to the theoretical maximum throughput, which is

$$\frac{1446}{1538} \cdot 1000\ Mb/s = 994\ Mb/s$$

We get the payload size of 1446 by removing all of the overhead that IPsec adds from the MTU of 1500 bytes. These headers include the IPv4 header (20 bytes) and the UDP header (8 bytes). It also includes the ESP-specific headers that were discussed in section 2.2.1. These include the SPI (4 bytes), sequence number (4 bytes), pad

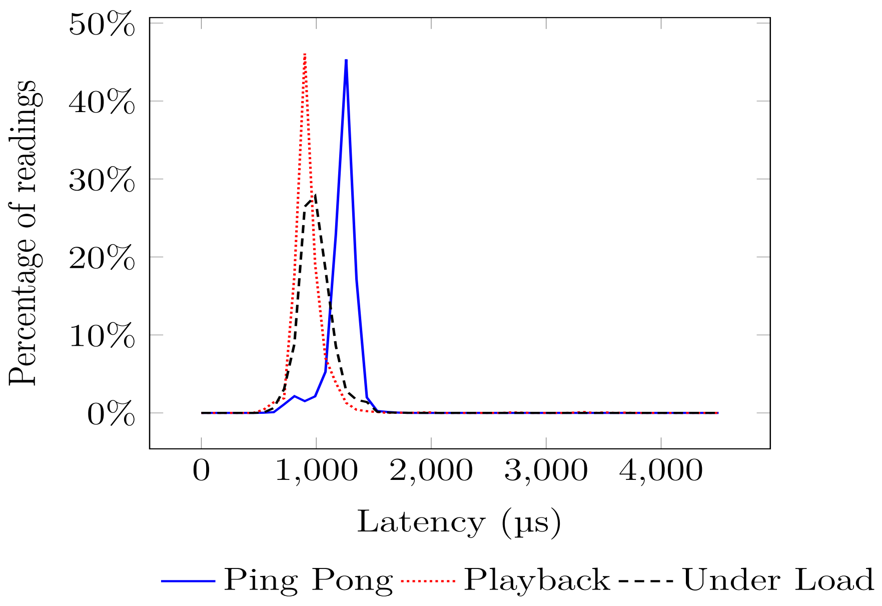

Figure 6.3: Latency distribution with IPsec enabled.

	Throughput
IPsec	938.5Mb/s

Table 6.4: IPsec throughput.

length (1 byte), the next header (1 byte), and finally, the ICV. The ICV depends on what integrity algorithm is used. In this test, SHA256 was used, which produces a 32-byte hash. However, the entire 32 bytes are not added to the packet. Instead, it is truncated down to 128 bits [59, Sec 5.3.3]. This brings the total available space down to 1446 bytes.

6.2.3 CPU Usage

The measurements from the CPU tests can be seen in Figure 6.4. Enabling IPsec on the system takes up a significant amount of resources from the CPU. The average CPU usage, excluding the first and last 10 seconds, is 7.3% of the total available resources. Most of these resources go to system-level calls, including hardware and software interrupts. This is expected as the part of IPsec is actually part of the Linux kernel, and the userspace tool is only there to handle the key setup and handling.

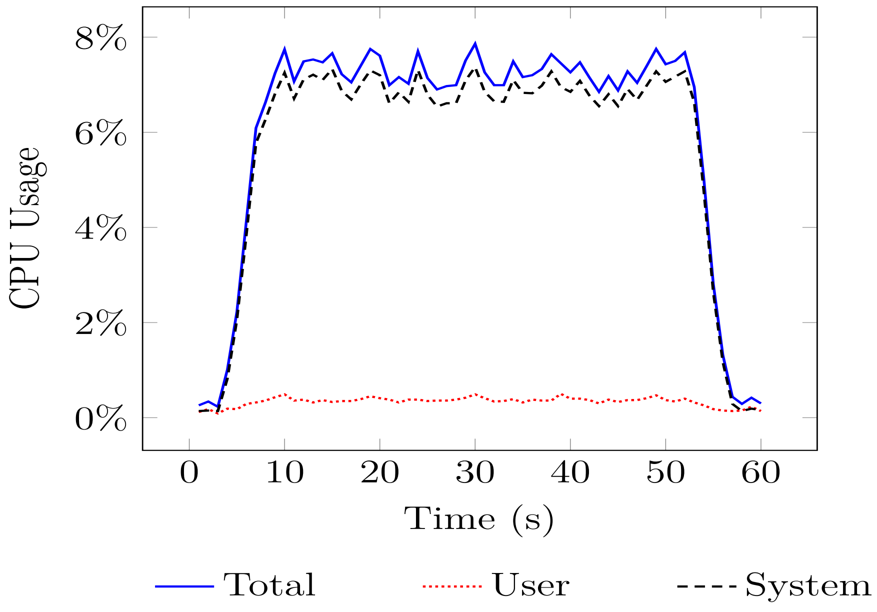

Figure 6.4: CPU usage with IPsec enabled.

6.3 MACsec

This section details the results from the tests done with MACsec configured on the network. The MACsec setup utilized pre-shared keys that were the same between all the nodes on the network.

6.3.1 Latency

Figure 6.5 shows the distribution of the latency of the measurements that were taken during the Ping Pong, Playback, and Under Load tests with MACsec configured on the network. The figure shows a small tail on all the tests. The tail is especially noticeable on the Under Load and Playback tests. However, it is uncertain from the figure if it surpasses the threshold of 2ms. The presence of a long-tail can be disproven by checking Table 6.5. While the latency varies quite a lot between the three different tests, none goes above the 2ms threshold.

(µs)	Ping Pong	Playback	Underload
90th	677	872	1025
99th	840	1272	1148
99.9th	942	1690	1201

Table 6.5: Maximum latency for the 90th, 99th, and 99.9th percentile with MACsec configured.

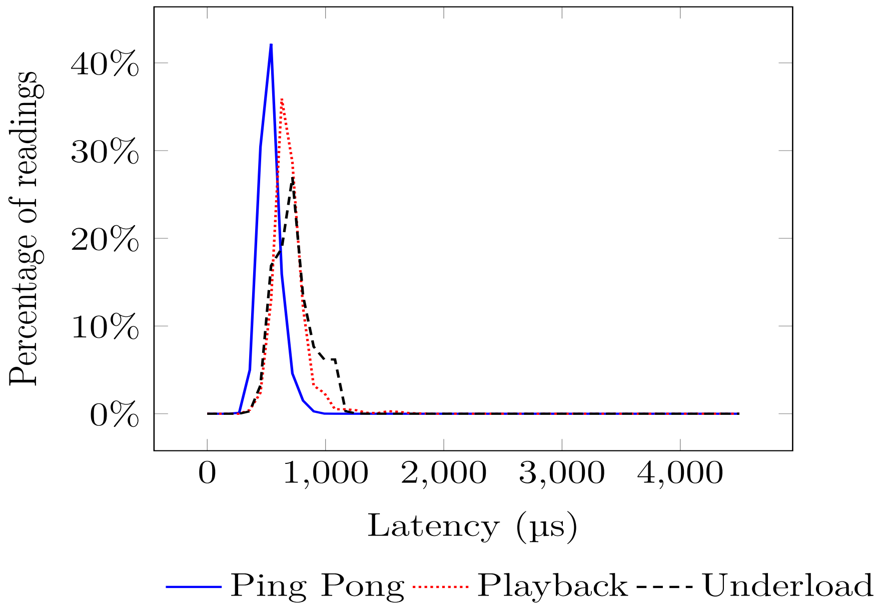

Figure 6.5: Latency distribution with MACsec enabled.

6.3.2 Throughput

The throughput test resulted in an average reading of 934.9 Mb/s, as seen in Table 6.6. This is close to the theoretical maximum for MACsec, which is calculated by

$$\frac{1440}{1538} \cdot 1000Mbps = 936.3\ Mbps.$$

This is a reduction of 2.3%, which is within the limit of a 5% throughput reduction.

	Throughput
MACsec	934.9Mb/s

Table 6.6: MACsec throughput.

6.3.3 CPU Usage

An average of the measurements taken during the CPU tests can be seen in Figure 6.6. It is immediately clear that enabling MACsec does incur a substantial load on the system. The average CPU usage, excluding the first and last 10 seconds, is 8.03% of the total available resources. Since MACsec is implemented into the Linux kernel, almost all of the CPU cycles used are from interrupt induces kernel-level processes.

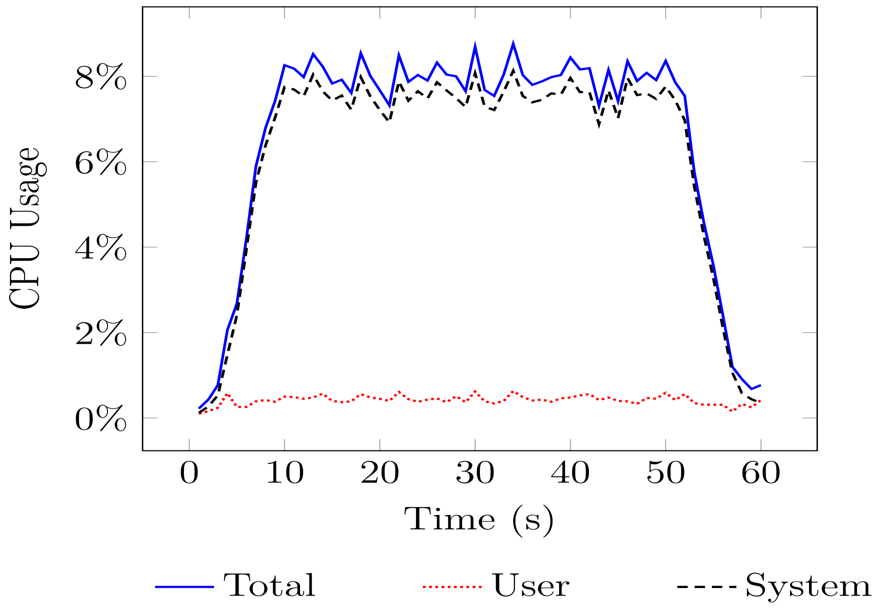

Figure 6.6: CPU usage with MACsec enabled.

6.4 Bump-in-the-Wire

This section details the results from the tests done with the MACsec enabled Bump-in-the-Wire device.

6.4.1 Latency

Figure 6.7 shows the distribution of the latency measurements that were taken during the Ping Pong, Playback, and Under Load tests with a MACsec enable Bump-in-the-Wire device. By looking at the graphs, it is clear that no apparent long-tail latency problem is present. This is further validated by Table 6.7 which shows that no latency in the 99.9th percentile goes above 1600 µs.

(µs)	Ping Pong	Playback	Underload
90th	1152	1233	1219
99th	1229	1397	1306
99.9th	1304	1535	1338

Table 6.7: Maximum latency for the 90th, 99th, and 99.9th percentile with a MACsec enabled Bump-in-the-wire.

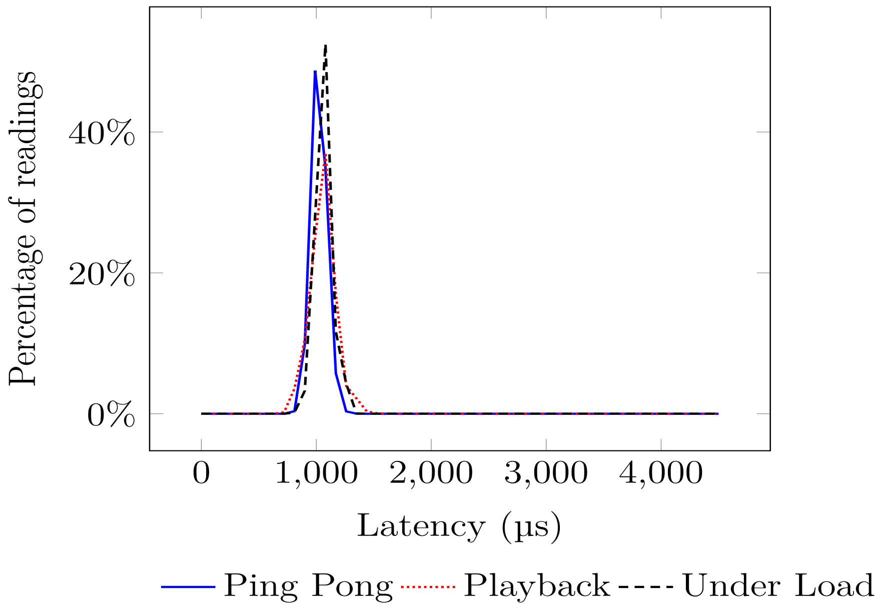

Figure 6.7: Latency distribution with a MACsec enabled Bump-in-the-Wire.

6.4.2 Throughput

The throughput tests resulted in a value of 911.9 Mb/s, as seen in Table 6.8. This is a 2.7% decrease compared to the theoretical maximum of 936.3 Mb/s.

	Throughput
Bump-in-the-Wire	911.9Mb/s

Table 6.8: Bump-in-the-Wire throughput.

6.5 Comparison

This section will compare the test results presented in sections 6.1, 6.2, and 6.3. Note that this section will use the Under Load tests when discussing latency. Graphs for comparing the results from Ping Pong and Playback can be found in Appendix A.

6.5.1 Latency

Figure 6.8 shows the distribution of the latency of the measurements for the four network configurations. The first observation that can be made is that MACsec generally has a lower latency than the network configured with IPsec, which in turn has a slightly lower latency then the Bump-in-the-Wire configuration. However, for the 99.9th percentile, which is the important metric in the context of this work, IPsec actually has a higher latency than the Bump-in-the-Wire, which can be seen in Table 6.9.

Furthermore, IPsec also has a more apparent long-tail than MACsec. While the long-tail is still within the accepted latency range, the difference between the median latency and the 99.9th percentile is 627 µs, compared to MACsec's difference of 454 µs. Interestingly, this can also be compared to the Bump-in-the-Wire configuration, which only has a difference of 229 µs.

(µs)	Baseline	IPsec	MACsec	Bump-in-the-Wire
90th	596	1212	1025	1219
99th	622	1468	1148	1306
99.9th	665	1621	1201	1338

Table 6.9: Maximum latency for the 90th, 99th, and 99.9th percentile for the four network configurations

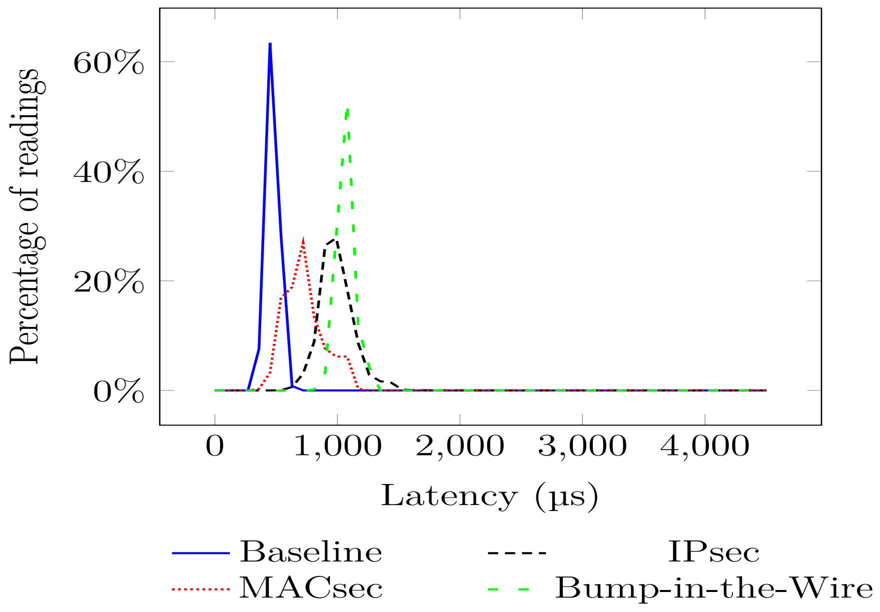

Figure 6.8: Latency distribution of the four network configurations.

6.5.2 Throughput

The throughput that was measured for the four network configurations can be seen in Figure 6.9. One comparison that can be made is to see if the different configurations do not reduce the throughput by more than five percent, as stipulated in section 5.1. The throughput will be compared to the baseline measurement of 956.1 Mb/s taken on the network that did not have any security configurations. With IPsec having a measured throughput of 938.5Mb/s, it reduces around 1.9% — thus clearing the requirement. The normal MACsec configuration had a measured throughput of 934.9 Mb/s, a reduction of 2.2%. Lastly, the Bump-in-the-Wire's throughput was measured at 911.9 Mb/s, a decrease of a little more than 4.6%. With this, all configurations clear the throughput threshold.

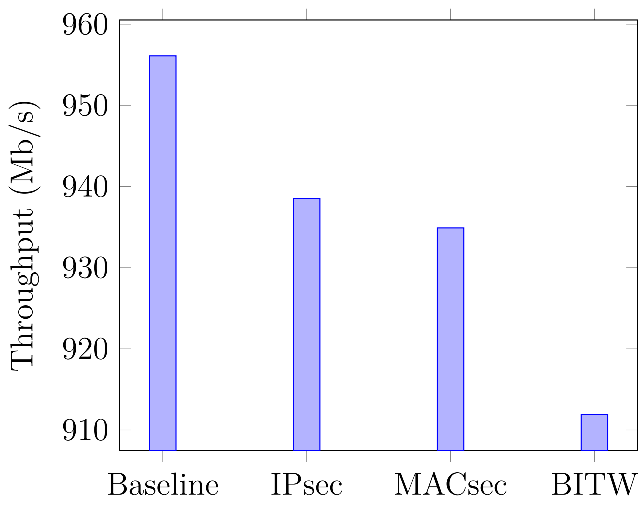

Figure 6.9: Throughput comparison of the four network configurations.

6.5.3 CPU Usage

Figure 6.10 shows the total CPU usage asserted on the consumer during stress-tests with the different network configurations. The tests were not done with the Bump-in-the-Wire device since there was not enough hardware to configure three devices with three corresponding sensors. However, the curve should be the same as the one with the "normal" MACsec configuration, considering that the consumer side of the connection is the same.

Unlike the Latency requirements, neither one of the two security protocols are within the threshold of 5%. MACsec uses 5.76% of the total available resources, and IPsec uses 5.03%. By the requirements stipulated in section 5.1, none of these two protocols are viable when run entirely in software.

6.6 Security

While an essential aspect of the testing, the security tests do not yield results that can readily be displayed. Instead, they lead to a binary yes or no regarding whether

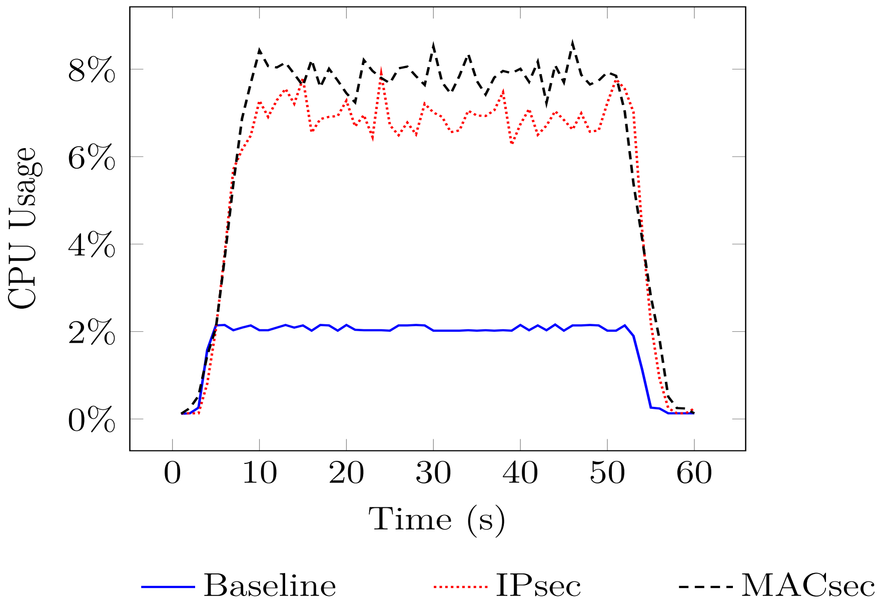

Figure 6.10: Comparison of CPU usage.

they work or not. In this case, the yes or no came in the form of the security protocol dropping the malicious message or not. The tests that were performed all resulted in the message being dropped, which means that the security guarantees described in their respective standards are present in the software implementations of the two protocols.

6.7 Key Exchange

The key exchange tests are meant to test the general viability of the setup and the support present in the current implementation of the different userspace programs used. The reason that the implementations have to be tested is because of a general lack of documentation. This shortage of documentation is not limited to wpa_supplicant and hostapd, but everything that relates to the software implementation of MACsec. Thus, the only way to find out if the implementations that handle the key distribution are viable is to test them.

As explained in section 5.3.2, the setup of the key exchange requires three parts: the supplicant, the authenticator, and the authentication server. wpa_supplicant was used as the supplicant program, hostapd as the authenticator, and FreeRADIUS as the authentication server. The exchange works by first having the supplicant, the wpa_supplicant program in the sensor or Bump-in-the-Wire device, attempt to connect to the authenticator. The authenticator will then contact the RADIUS server that will either deny or accept the authentication request. If the authentication is successful, the authenticator and the supplicant will set up a MACsec secured connection.

The authentication part of this test worked as expected. The supplicant successfully authenticated to the RADIUS server when the server had a server certificate, the sensor had a client certificate, and the same certificate authority signed them both. However, hostapd was then unable to create the MACsec interface. The error that caused the interface creation to fail was that hostapd could not find the session ID in the EAPOL state machine. It is possible that this could have been due to a configuration error, but no obvious answer to what that error could have been was found.

The authentication and MACsec interface creation were successfully set up by utilizing an older and modified version of hostapd[1]. While unsuitable for production, it does show that the key exchange can work in practice. Another successfully tested solution was to only utilize two running wpa_supplincat instances that authenticate with each other with a preshaped CAK and CKN. While it does not use the certificate-based design suggested in this work, it does show that key distribution and generation work with pre-shared keys.

While testing the two working setups, it was also discovered that neither hostapd nor wpa_supplicant can create two MACsec interfaces on the same physical device. The working hostapd implementation kept re-authenticating the two devices, setting up a new interface every time a new device got authenticated and deauthenticated the old device. wpa_supplicant would throw an error and stop working for all connected devices. There is also no indication that groups SAKs are implemented in the Linux version of MACsec, which would let multiple devices communicate with the same MACsec interface. For the key exchange to work, a virtual ethernet interface would have to be set up for each sensor that is supposed to connect to a specific port.

[1]`https://github.com/alessandrotrisolini/hostapd`

7

Discussion

This chapter discusses topics that are relevant to the conclusion of the thesis. Sections 7.1, 7.2, and 7.3 relates the research questions posed in section 1.2 to the results detailed in chapter 6. Additionally, section 7.3 talk about a possible production setup for the distribution of certificates.

7.1 Research Question 1

The first research question was: Can protocols ensuring message integrity be effectively run on software with regards to performance? Two different protocols were tested, one that secures IP traffic and one that secures Ethernet traffic. The two tested solutions, IPsec and MACsec, were chosen because they are both standardized either by IEEE or as an RFC. The protocols were then tested based on three criteria: latency, throughput, and CPU utilization. This section is structured around the findings, and a conclusion is summarized at the end.

7.1.1 Latency

For the particular environment that this thesis is concerned about, the internal network of an autonomous vehicle, the latency requirement is one of the most important. While throughput or CPU utilization is more of a problem of resource allocation, how much of the computing power do you want to designate to security, a high latency would render the software solution ineffective. Fortunately, the test showed that all network configurations work with a latency under 2 ms for the 99.9th percentile. Both IPsec, MACsec, and the Bump-in-the-Wire device are legitimate solutions for securing the network with regard to latency.

There are differences in the performance between the setups. The first and most obvious conclusion that can be drawn is that MACsec is superior to IPsec in regards to latency. There are multiple reasons for this. Firstly, the algorithm used for

MACsec, GCM-AES-128, is significantly faster than the one used for IPSec, SHA256. The full reason for this is too long to cover in this thesis, but one point can be raised. Both SHA256 and GCM-AES-128 use blocks of data to calculate their hash. However, while SHA256 calculates the hash sequentially over the block, meaning that block n has to be calculated before block $n + 1$, GCM-AES-128 can do this in parallel - speeding up the process significantly.

Another reason why MACsec is quicker than IPsec is that MACsec adds authentication on Ethernet packets and not IP packets. The tests on latency used packets of size 24938 bytes, which causes the packet to fragment since the MTU is set to 1500. MACsec adds an ICV to every one of these fragments, while IPsec only adds one for the entire un-fragmented IP packet. This allows MACsec to check the integrity whenever a fragment arrives, while IPsec has to wait for all fragments to arrive and then perform the reassembly before the integrity check can be done.

One thing that should be mentioned is that while Chapter 6 presented results down to a microsecond accuracy, this type of accuracy is not something that can be expected. For instance, while the Under Load measurement came out to 1201 µs for MACsec, this does not mean that the highest latency will be 1201 µs. Instead, it gives an indication of where the highest latency might end up, or what latency it will not surpass. In this case, the highest latency might be in the 1150 µs to the 1250 µs range, and the it can be assumed that the latency will not surpass 1300 µs — or more importantly 2 ms.

7.1.2 Throughput

While the throughput can not be guaranteed to exactly match the results in Chapter 6, the results do indicate that the throughput will not be reduced by more than 5%. However, something should be said about how throughput will change depending on packet size. For IPsec, the added overhead changed dramatically based on the original size of the IP packet. According to the RFC 2406, fragmentation should always be done after ESP processing[57]. Therefore, the IV will only be added to the original sized IP packet, not the individual fragments.

For instance, the IPsec specific overhead added to the sensor simulation packet is only 26 bytes. Since the original packet is 24938 bytes, the IPsec overhead only accounts for 0.1% of the total packet. If the packet instead were to get smaller, for instance, 64 bytes, the throughput would significantly degrade. This time, 28 bytes will be added, and two bytes will be added to the payload as padding to align the Next and SCI fields to the right. In this scenario, the IPsec overhead will account for a 43% increase in packet size, reducing the efficiency to 69.5%.

As MACsec is added onto the Ethernet frames, the largest packet that can be sent is 1500 bytes - assuming that jumbo frames are not supported. Thus, the highest throughput that can be registered is the same as the one registered in section 6.3.2. If the packet size were to be reduced to 64 bytes, the MACsec overhead would lead to a 50% increase in packet size. This can also be seen as a reduction in efficiency to 67%. Sensors that send smaller packets, for instance, GPS, will significantly reduce in efficiency as most of the packet will be the MACsec header. While not detrimental to the system's functionality, the network setup might have to be reconfigured to account for the added overhead.

7.1.3 CPU Usage

The CPU usage is the biggest resource sink for both the security protocols. While both the latency and the throughput were within the requirements stipulated in section 5.1, the CPU usage was not. IPsec took up between 4-6% of the total CPU resources, and MACsec used between 5-7%. This is above the allowed 5% allocation of the total CPU resources available.

While this would disqualify both of these protocols from being viable in an in-vehicle network, it should be noted that this 5% limit is somewhat arbitrary. Multiple factors have to be taken into account when deciding on the amount of CPU resources that can be dedicated to the security of the network. For instance, if there is an abundance of CPU resources left continuously idle, there is no reason that these cannot be used for security protocols. Furthermore, if the vehicle is operating in an especially exposed environment, security might have to be prioritized.

One thing that should be brought up is that the amount of resources used by the security protocols seems to have a somewhat linear correlation with the amount of data the consumer has to process. While this was not explicitly tested, which is why it is not mentioned in section 6, it could be seen while the tests were set up. So, while the recorded CPU loads were just over the threshold, this would not be the case if more sensors were added. Every time more sensors would be added, the CPU usage would increase, and the viability of the current security setup would have to be reevaluated.

With the requirements set up in section 5.1, the security protocols can not be seen as viable when run entirely in software. Instead, it is up to the manufacturers to decide if an additional increase of 2% in CPU usage is warranted for the security. If it is, both the protocols are viable options when run in software. However, it is still advisable to add hardware acceleration when implementing the security protocols. Hardware acceleration would reduce CPU usage significantly. Folkemark and Rydberg's master thesis has shown that an HSM can offload up to 94% on a GCM-based cipher[60]. This would make the protocols more viable for the current setup and when adding more sensors in the future.

7.1.4 Summary

To summarize, while both the latency and throughput requirements were met, the CPU requirements were not. Both IPsec and MACsec required slightly more than 5% of the total CPU resource to operate. Considering this, it would be advisable to utilize some sort of hardware acceleration to offload the CPU from the cryptographic calculations.

7.2 Research Question 2

The second research question was: Can intermediate devices be used to secure unsecured devices? More precisely, the goal was to see if an intermediate device could be introduced and still adhere to the performance requirements in Research Question One. Due to time limitations, an intermediate device was only created for MACsec. The reason that MACsec was chosen over IPsec was that the theoretical study done before any of the testings suggested that MACsec would produce good results when it came to latency.

The implementation itself turned out to work well with regard to performance. The latency was not increased substantially, going from a median latency of 747.6µs to 1109 µs. This still leaves a buffer where the latency could increase by another 45% and not break the requirements set up. Furthermore, ignoring the 0.1% of top outliers, the latency never went above 1340.1µs. So for the latency requirements, the intermediate device worked well. Furthermore, since the latency did not increase significantly, throughput only slightly decreased.

The reason that the latency did not increase more than it did is most likely because the intermediate device could be implemented by only utilizing a network bridge. This avoids the risk of the device having to handle any fragmentation, which could be a source of both latency and CPU usage.

While the intermediate device solution theoretically is suitable for retrofitting security protocols onto sensors, it has drawbacks. First of all, it will increase the number of potential points of failure in the network. If one of the intermediary devices were to break down, that would cause the sensor to lose connection to the main computer, possibly taking the vehicle out of service. However, these devices are not too complicated, thus decreasing its risk.

Another downside is that the hardware for the intermediate device most likely has to be custom-made. While smaller computers will work as a medium, such as the device used in this work, these types of consumer devices are not made for or are likely to survive the harsh environments of a vehicle. Furthermore, the device will also need to be equipped with two Ethernet ports that support automotive ethernet.

The final downside is that the verification is not done with the sensor but with the intermediate device. This could result in a malicious actor disconnecting the sensor and plugging in a malicious device into the intermediate, thus masquerading as a verified device. Therefore, the intermediate device will have to be securely attached to the sensor, forcing the construction of purpose-made hardware.

7.2.1 Summary

The purpose of this question was to look at one way of retrofitting security into previously unsecured nodes. While the performance measurements show that the solution is viable, it has some significant practical downsides. However, these downsides will only incur a one-time cost, after which all nodes on the Ethernet part of the IVN can be secured.

7.3 Research Question 3

The last question was: How should keys be generated, distributed, and stored in an in-vehicle network? This question was more theoretical than practical. The aim was to give a suggestion on how the keys could be handled if one of the security protocols were to be adopted into the IVN. Some conditions were that the key exchange should not be proprietary, as that would make the key exchange incompatible with any sensor that might incorporate one of the security protocols.

7.3.1 Pre-Shared Keys

One way that the key exchange could be done is by utilizing pre-shared keys (PSK). The PSKs used in this scenario are meant for authentication, not for the communication itself. One of the upsides of utilizing a pre-shared key is that it is easy to configure. Both parties simply need to share the key. In MACsec, this comes in the form of a Connectivity Association Key and a Connectivity Association Key Name that both the sensor and main computer have access to. In IPsec, a unique key is set up for every IP pair that is supposed to be secured.

However, while it is easy to set up and convenient in a small network that does not change often, it is not scalable. As soon as a sensor has to be changed, new keys have to be generated and distributed to all affected parties. The old keys also have to be removed from the old sources. Furthermore, pre-shared keys do not have security measures such as expiration dates or identifying information.

7.3.2 Certificates

Another way that keys can be exchanged is by utilizing certificates. This configuration requires three parties: a certificate authority, a server certificate, and one or more client certificates. The way that a system using certificates works compared to PSK is that the root of trust is moved from the knowledge of what the PSK is to the certificate authority. In a PSK system, if the device knows the key, the other devices will trust that the knowledge of the key was acquired legitimately. In other words, the devices have to trust the devices. In a certificate-based system, the devices don't have to trust each other.

Each sensor will be given a client certificate and the main computer a server certificate. These certificates have been generated by the certificate authority. The certificate authority has a private key used to sign new certificates. It also has a public certificate that can verify the certificates that it signs. Thus, when a client presents a server with its certificate, the server can verify that the CA created the certificate. Because the server trusts the CA, it will also trust the client. This procedure is also done by the client and the server.

7.3.3 Summary

Two different solutions on how to handle the key generation, distribution, and storage have been proposed. One is based on utilizing pre-shared keys, while the other uses digital certificates. While both techniques are viable solutions, the one based on digital certificates is more scalable and more in line with the zero trust model. Since it utilizes a RADIUS server for authentication and authorization, the access given to the connecting devices can be better controlled.

8 Conclusion

The goal of this work was to present a possible solution for moving the internal vehicle network from relying on a perimeter security model to a zero trust model. The general idea of how this transformation was going to take place is by implementing message integrity, letting the main computer verify every message that comes in. Although zero trust generally includes confidentiality as well as integrity, the resources are currently not available for encrypting the amount of data that is being transferred on the internal network.

The thesis divided the introduction of a zero trust architecture into three different questions. The first question looked into the performance impact that a software solution to message integrity might have. Two different protocols, IPsec and MACsec, that guarantee message integrity was analyzed and compared against a baseline reading of the network without any security configurations. It was found that both protocols perform well enough to be integrated into an in-vehicle network. However, MACsec did consume more CPU resources than was required. If resources are limited in the main computer, this could prevent the adoption of a software-based version of MACsec. However, it did perform a lot better than IPsec in regards to latency.

The second area that was looked into was how to secure nodes that do not support any security protocol. The suggested solution was to implement a so-called ”Bump-in-the-Wire” device that transparently adds the security fields to any traffic that flows through it. Due to time constraints, a Bump-in-the-wire were only constructed for MACsec. It utilizes a Linux network bridge to transparently forward any ethernet traffic. Due to MACsec encrypting ethernet traffic, no resources have to be spent on calls other than securing the actual traffic. The Bump-in-the-wire thus performed very well in all regards, even outperforming IPsec in terms of latency. Furthermore, since the construction is so simple it does not need a lot of resources to perform effectively.

The last area that was looked into is how the key distribution should be handled for the network. Compared to the other two areas that focused on the performance, this area simply should present a solution for the distribution of keys. The suggested solution is based on 802.1X authentication with certificates using RADIUS as the authentication server.

To conclude, this thesis has presented a verified and performance-checked way of implementing a zero trust architecture into the in-vehicle network. However, if possible, it is still preferable to implement it with a hardware version of MACsec instead of one that is implemented in software. Furthermore, it would also be preferable to have integrated security into every sensor instead of relying on Bump-in-the-Wire devices. With these two changes, and in combination of the suggested key distribution scheme, a secure network has been created.

8.1 Future Work

This section presents some ideas for future work within this field.

8.1.1 Hardware offloading for MACsec

One area that could have a huge impact on the performance, and consequently viability, of security in in-vehicle networks is cryptographic hardware acceleration. While most CPUs come with some form of cryptographic acceleration, for instance, AES-NI in Intel chips, there are some more purposefully built. While the selection of MACsec enabled PHYs is quite limited there are some being produced by Microsemi, as well as Intel's ixgbe NIC. Support for offloading MACsec operations to these drivers was introduced into the MACsec module in January of 2020. A study into the performance benefits of offloading the MACsec operations might give an indication of the viability of implementing the protocol in an automotive setting.

There are also some PHYs sold that have a complete hardware implementation of MACsec. These PHYs can encrypt traffic at wire speed, thus not affecting the network any capacity except for throughput.

8.1.2 MACsec Key Agreement

The tests meant to analyze the implementation of MKA in wpa_supplicant and hostapd were not successful. A working setup was never created when using the latest release of hostapd. Instead, an older and modified release had to be used. Furthermore, wpa_supplicant could only establish a MACsec channel when utalizing pre-shared keys and not with digital certificates. For the solution to research question three to be viable, some more research either has to go into testing the wpa_supplicant and hostapd programs more or implement a new program that can handle MKA.

Bibliography

[1] A. Greenberg, “Hackers remotely kill a jeep on the highway—with me in it,” *Wired*. [Online]. Available: https://www.wired.com/2015/07/hackers-remotely-kill-jeep-highway/

[2] ——, “This gadget hacks gm cars to locate, unlock, and start them,” *Wired*. [Online]. Available: https://www.wired.com/2015/07/gadget-hacks-gm-cars-locate-unlock-start/

[3] C. Hatipoglu. (2021) Cybersecurity best practices for the safety of modern vehicles. [Online]. Available: https://www.nhtsa.gov/sites/nhtsa.gov/files/documents/cbpsmv_federal_register_notice_01072021_0.pdf

[4] “Proposal for amendments to ece/trans/wp.29/grva/2020/3,” Tech. Rep. [Online]. Available: https://unece.org/fileadmin/DAM/trans/doc/2020/wp29grva/GRVA-06-19r1e.pdf

[5] N. Nowdehi, “Automotive Communication Security methods and recommendations for securing in-vehicle and v2x communications,” 2019.

[6] “Ieee standard for ethernet amendment 1: Physical layer specifications and management parameters for 100 mb/s operation over a single balanced twisted pair cable (100base-t1),” *IEEE Std 802.3bw-2015 (Amendment to IEEE Std 802.3-2015*, pp. 1–88, 2016.

[7] Aptiv PLC. (2020) What is an electronic control unit? Accessed: 2022-05-23. [Online]. Available: https://www.aptiv.com/en/insights/article/what-is-an-electronic-control-unit

[8] K. Ç. Bayindir, M. A. Gözüküçük, and A. Teke, “A comprehensive overview of hybrid electric vehicle: Powertrain configurations, powertrain control techniques and electronic control units,” *Energy conversion and Management*, vol. 52, no. 2, pp. 1305–1313, 2011.

[9] Y. Cao, T. Griffon, F. Fahrenkrog, M. Schneider, F. Naujoks, F. Tango, S. Wolter, A. Knapp, Y. Page, J. Mallada, G. Dominioni, E. Demirtzis,

M. Giorelli, S. Fabello, F. Frey, Q. Feng, O. Brunnegård, A. Kucewicz, S. Whitehouse, and U. Eberle, "L3pilot - code of practice for the development of automated driving functions," Tech. Rep., 04 2021.

[10] F. Götz, "The data deluge: What do we do with the data generated by avs?" *Siemens Blog*. [Online]. Available: https://blogs.sw.siemens.com/polarion/the-data-deluge-what-do-we-do-with-the-data-generated-by-avs/

[11] C. E. Mika Arpe. Why automotive ethernet is a strong choice. [Online]. Available: https://www.aptiv.com/en/insights/article/why-automotive-ethernet-is-a-strong-choice

[12] Research and Markets, "Global automotive ethernet market report and forecast 2021-2026," 10 2021. [Online]. Available: https://www.researchandmarkets.com/r/2mjhvw

[13] J. Z. Varghese, R. G. Boone *et al.*, "Overview of autonomous vehicle sensors and systems," in *International Conference on Operations Excellence and Service Engineering*. sn, 2015, pp. 178–191.

[14] I. Bilik, O. Longman, S. Villeval, and J. Tabrikian, "The rise of radar for autonomous vehicles: Signal processing solutions and future research directions," *IEEE Signal Processing Magazine*, vol. 36, no. 5, pp. 20–31, 2019.

[15] J. Kocić, N. Jovičić, and V. Drndarević, "Sensors and sensor fusion in autonomous vehicles," in *2018 26th Telecommunications Forum (TELFOR)*, 2018, pp. 420–425.

[16] Y. Li and J. Ibanez-Guzman, "Lidar for autonomous driving: The principles, challenges, and trends for automotive lidar and perception systems," *IEEE Signal Processing Magazine*, vol. 37, no. 4, pp. 50–61, 2020.

[17] A. Zaarane, I. Slimani, W. Al Okaishi, I. Atouf, and A. Hamdoun, "Distance measurement system for autonomous vehicles using stereo camera," *Array*, vol. 5, p. 100016, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2590005620300011

[18] S. Tuohy, M. Glavin, C. Hughes, E. Jones, M. Trivedi, and L. Kilmartin, "Intra-vehicle networks: A review," *IEEE Transactions on Intelligent Transportation Systems*, vol. 16, no. 2, pp. 534–545, 2015.

[19] J. F. Kurose and K. W. Ross, *Computer Networking: A Top-Down Approach*, 7th ed. Boston, MA: Pearson, 2016.

[20] W. Stallings, *Cryptography and Network Security: Principles and Practice*, 6th ed. USA: Prentice Hall Press, 2013.

[21] "Ieee standard for local and metropolitan area networks-media access control (mac) security," *IEEE Std 802.1AE-2018 (Revision of IEEE Std 802.1AE-2006)*, pp. 1–239, 2018.

[22] "Ieee standard for local and metropolitan area networks-media access control (mac) security," *IEEE Std 802.1AE-2018 (Revision of IEEE Std 802.1AE-2006)*, pp. 1–239, 2018.

[23] Z. Abdin, "Protecting integrity and confidentiality of network traffic with media access control security (macsec)," 2021.

[24] N. Χαλβατζή, "Analysis and empirical evaluation of ieee 802.1 ae (mac security) standard," 2020.

[25] S. Frankel and S. Krishnan, "IP Security (IPsec) and Internet Key Exchange (IKE) Document Roadmap," RFC 6071, Feb. 2011. [Online]. Available: https://www.rfc-editor.org/info/rfc6071

[26] C. Kaufman, P. E. Hoffman, Y. Nir, P. Eronen, and T. Kivinen, "Internet Key Exchange Protocol Version 2 (IKEv2)," RFC 7296, Oct. 2014. [Online]. Available: https://www.rfc-editor.org/info/rfc7296

[27] W. Diffie and M. E. Hellman, "New directions in cryptography," in *Secure communications and asymmetric cryptosystems.* Routledge, 2019, pp. 143–180.

[28] "Ieee standard for local and metropolitan area networks – port-based network access control amendment 1: Mac security key agreement protocol (mka) extensions," *IEEE Std 802.1Xbx-2014 (Amendment to IEEE Std 802.1X-2010)*, pp. 1–107, 2014.

[29] J. Vollbrecht, J. D. Carlson, L. Blunk, D. B. D. Aboba, and H. Levkowetz, "Extensible Authentication Protocol (EAP)," RFC 3748, Jun. 2004. [Online]. Available: https://www.rfc-editor.org/info/rfc3748

[30] D. Simon, R. Hurst, and D. B. D. Aboba, "The EAP-TLS Authentication Protocol," RFC 5216, Mar. 2008. [Online]. Available: https://www.rfc-editor.org/info/rfc5216

[31] J. Daemen and V. Rijmen, "Aes proposal: Rijndael," 1999.

[32] D. A. McGrew, "Counter mode security: Analysis and recommendations," *Cisco Systems, November*, vol. 2, no. 4, 2002.

[33] D. McGrew and J. Viega, "The galois/counter mode of operation (gcm)," *submission to NIST Modes of Operation Process*, vol. 20, pp. 0278–0070, 2004.

[34] S. D. Young, “Moving the u.s. government toward zero trust cybersecurity principles,” official memorandum, Executive Office of the President, Washington, DC, USA, Washington, DC: USA, 20122 [Online]. [Online]. Available: https://www.whitehouse.gov/wp-content/uploads/2022/01/M-22-09.pdf

[35] *Zero Trust Reference Architecture*, Department of Defense, Washington, DC, USA, 2021. [Online]. Available: https://dodcio.defense.gov/Portals/0/Documents/Library/(U)ZT_RA_v1.1(U)_Mar21.pdf

[36] A. Karahasanovic, P. Kleberger, and M. Almgren, in *Adapting Threat Modeling Methods for the Automotive Industry*, 2017.

[37] B. Parno and A. Perrig, “Challenges in securing vehicular networks,” 01 2005.

[38] F. Sommer, J. Dürrwang, and R. Kriesten, “Survey and classification of automotive security attacks,” *Information*, vol. 10, no. 4, 2019. [Online]. Available: https://www.mdpi.com/2078-2489/10/4/148

[39] L. Liu, S. Lu, R. Zhong, W. Baofu, Y. Yao, Q. Zhang, and W. Shi, “Computing systems for autonomous driving: State-of-the-art and challenges,” *IEEE Internet of Things Journal*, vol. PP, pp. 1–1, 12 2020.

[40] C. Corbett, E. Schoch, F. Kargl, and F. Preussner, “Automotive ethernet: security opportunity or challenge?” in *Sicherheit 2016 - Sicherheit, Schutz und Zuverlässigkeit*, M. Meier, D. Reinhardt, and S. Wendzel, Eds. Bonn: Gesellschaft für Informatik e.V., 2016, pp. 45–54.

[41] F. Sagstetter, M. Lukasiewycz, S. Steinhorst, M. Wolf, A. Bouard, W. R. Harris, S. Jha, T. Peyrin, A. Poschmann, and S. Chakraborty, “Security challenges in automotive hardware/software architecture design,” in *2013 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2013, pp. 458–463.

[42] H. Ju, B. Jeon, D. Kim, B. Jung, and K. Jung, “Security considerations for in-vehicle secure communication,” in *2019 International Conference on Information and Communication Technology Convergence (ICTC)*, 2019, pp. 1404–1406.

[43] A. Talic, “Security analysis of ethernet in cars,” 2017.

[44] M. Lang, “Secure automotive ethernet: Balancing security and safety in time sensitive systems,” 2019.

[45] J. Lastinec and L. Hudec, “Comparative analysis of tcp/ip security protocols for use in vehicle communication,” in *2016 17th International Carpathian Control Conference (ICCC)*, 2016, pp. 429–433.

[46] T. Lauser, D. Zelle, and C. Krauß, "Security analysis of automotive protocols," in *Computer Science in Cars Symposium*, ser. CSCS '20. New York, NY, USA: Association for Computing Machinery, 2020. [Online]. Available: https://doi.org/10.1145/3385958.3430482

[47] H.-Y. Lee and D.-H. Lee, "Security of ethernet in automotive electric/electronic architectures," *The Journal of the Institute of Internet Broadcasting and Communication*, vol. 16, pp. 39–48, 10 2016.

[48] J.-H. Choi, S.-G. Min, and Y.-H. Han, "Macsec extension over software-defined networks for in-vehicle secure communication," in *2018 Tenth International Conference on Ubiquitous and Future Networks (ICUFN)*, 2018, pp. 180–185.

[49] B. Carnevale, F. Falaschi, L. Crocetti, H. Hunjan, S. Bisase, and L. Fanucci, "An implementation of the 802.1ae mac security standard for in-car networks," in *2015 IEEE 2nd World Forum on Internet of Things (WF-IoT)*, 2015, pp. 24–28.

[50] K.-S. Han, K.-O. Kim, T. W. Yoo, and Y. Kwon, "The design and implementation of mac security in epon," in *2006 8th International Conference Advanced Communication Technology*, vol. 3, 2006, pp. 4 pp.–1676.

[51] T. Lackorzynski, G. Garten, J. S. Huster, S. Köpsell, and H. Härtig, "Enabling and optimizing macsec for industrial environments," *IEEE Transactions on Industrial Informatics*, vol. 17, no. 11, pp. 7599–7606, 2021.

[52] Y. Nir and A. Langley, "ChaCha20 and Poly1305 for IETF Protocols," RFC 7539, May 2015. [Online]. Available: https://www.rfc-editor.org/info/rfc7539

[53] A. Shriram, M. Murray, Y. Hyun, N. Brownlee, A. Broido, M. Fomenkov *et al.*, "Comparison of public end-to-end bandwidth estimation tools on high-speed links," in *International Workshop on Passive and Active Network Measurement.* Springer, 2005, pp. 306–320.

[54] C. Schroder, "Measure network performance with iperf," *Enterprise Networking Planet*, 2007.

[55] D. Yin, E. Yildirim, S. Kulasekaran, B. Ross, and T. Kosar, "A data throughput prediction and optimization service for widely distributed many-task computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 22, no. 6, pp. 899–909, 2010.

[56] E. Yildirim, I. H. Suslu, and T. Kosar, "Which network measurement tool is right for you? a multidimensional comparison study," in *2008 9th IEEE/ACM International Conference on Grid Computing.* IEEE, 2008, pp. 266–275.

[57] S. Kent and R. Atkinson, “IP Encapsulating Security Payload (ESP),” RFC 2406, Nov. 1998. [Online]. Available: https://www.rfc-editor.org/info/rfc2406

[58] S. Kent, “Privacy Enhancement for Internet Electronic Mail: Part II: Certificate-Based Key Management,” RFC 1422, Feb. 1993. [Online]. Available: https://www.rfc-editor.org/info/rfc1422

[59] S. Frankel and S. G. Kelly, “Using HMAC-SHA-256, HMAC-SHA-384, and HMAC-SHA-512 with IPsec,” RFC 4868, May 2007. [Online]. Available: https://www.rfc-editor.org/info/rfc4868

[60] M. Folkemark and V. Rydberg, “Performance evaluation of a hardware security module in vehicles,” Master’s thesis, Chalmers University of Technology, 2021.

A

Results

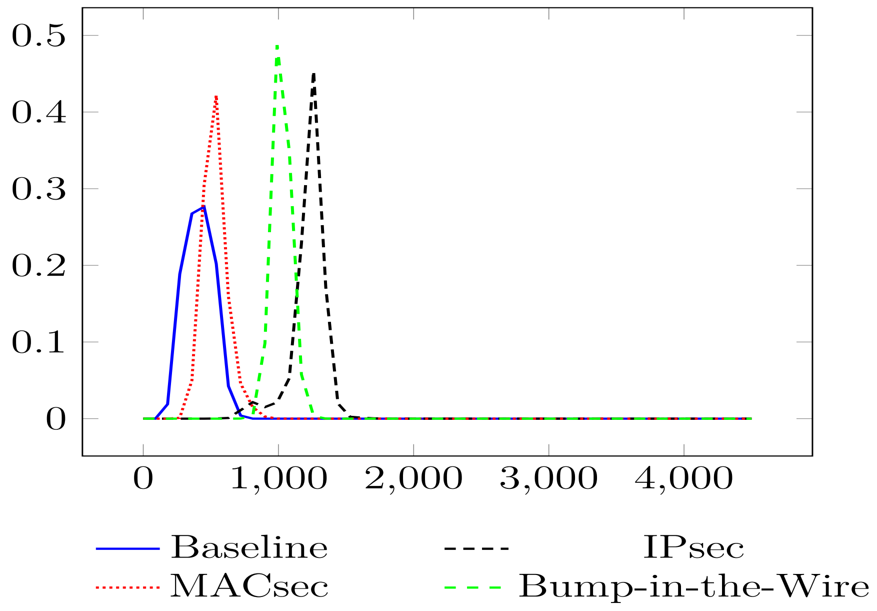

Figure A.1: Latency distribution of the four network configurations after the Ping Pong test.

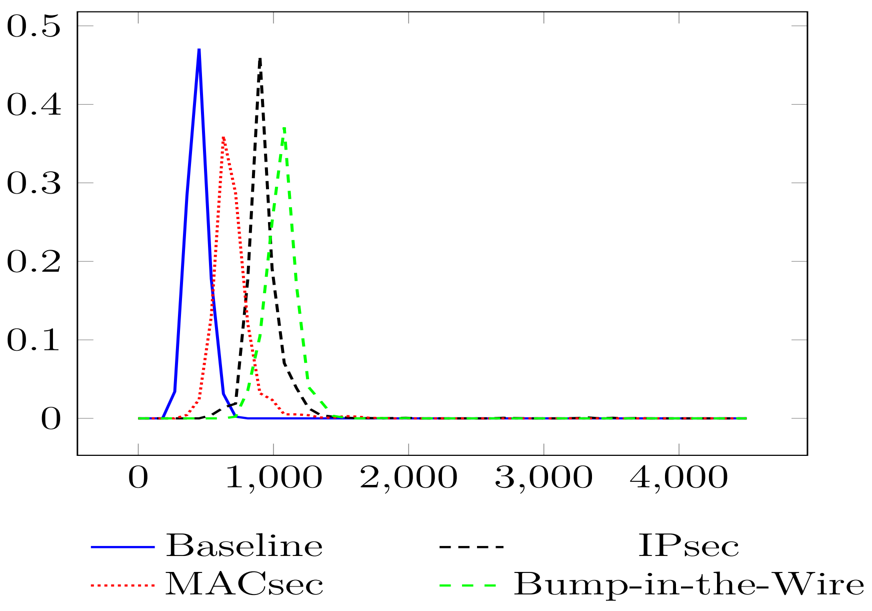

Figure A.2: Latency distribution of the four network configurations after the Playback test.