Informatics

Eignung von Hardware Sicherheitsmodulen für die Absicherung von Remote Zugängen in Hochleistungsumgebungen

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Software Engineering & Internet Computing

eingereicht von

Stephan Goldschmidt, BSc
Matrikelnummer 0625763

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Thomas Grechenig
Mitwirkung: Florian Fankhauser

Wien, 24. März 2025

Unterschrift Verfasser

Unterschrift Betreuung

Informatics

Suitability of Hardware Security Modules for Securing Remote Access in High-Performance Environments

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Software Engineering & Internet Computing

by

Stephan Goldschmidt, BSc
Registration Number 0625763

to the Faculty of Informatics

at the TU Wien

Advisor: Thomas Grechenig
Assistance: Florian Fankhauser

Vienna, 24th March, 2025

Signature Author | Signature Advisor

Eignung von Hardware Sicherheitsmodulen für die Absicherung von Remote Zugängen in Hochleistungsumgebungen

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Software Engineering & Internet Computing

eingereicht von

Stephan Goldschmidt, BSc
Matrikelnummer 0625763

ausgeführt am
Institut für Information Systems Engineering
Forschungsbereich Business Informatics
Forschungsgruppe Industrielle Software
der Fakultät für Informatik der Technischen Universität Wien

Betreuung: Thomas Grechenig

Wien, 24. März 2025

Erklärung zur Verfassung der Arbeit

Stephan Goldschmidt, BSc

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe. Ich erkläre weiters, dass ich mich generativer KI-Tools lediglich als Hilfsmittel bedient habe und in der vorliegenden Arbeit mein gestalterischer Einfluss überwiegt. Im Anhang „Übersicht verwendeter Hilfsmittel" habe ich alle generativen KI-Tools gelistet, die verwendet wurden, und angegeben, wo und wie sie verwendet wurden. Für Textpassagen, die ohne substantielle Änderungen übernommen wurden, haben ich jeweils die von mir formulierten Eingaben (Prompts) und die verwendete IT-Anwendung mit ihrem Produktnamen und Versionsnummer/Datum angegeben.

Wien, 24. März 2025

Stephan Goldschmidt

Kurzfassung

Ein virtuelles privates Netzwerk (VPN) ermöglicht den verschlüsselten Zugriff auf private Netzwerke. In Umgebungen mit besonders hohen Anforderungen an die Sicherheit kann es notwendig sein, die geheimen Schlüssel, die für den Betrieb von VPNs erforderlich sind, in besonderem Maße vor physischen Zugriffen zu schützen. Durch das Speichern der geheimen Schlüssel in einem Hardware-Sicherheitsmodul (HSM) können diese zwar vor physischen Angriffen geschützt werden, jedoch müssen alle kryptografischen Operationen, die diese Schlüssel nutzen, innerhalb des HSM durchgeführt werden. Dies kann je nach Einsatzumgebung bedeuten, dass im HSM wenige Hundert bis zu einigen Tausend Berechnungen pro Sekunde durchgeführt werden müssen. Diese Diplomarbeit untersucht die Eignung von HSMs zur physischen Absicherung von VPNs in Hochleistungsumgebungen.

Im Rahmen dieser Diplomarbeit wurde ein Prototyp einer Leistungstestumgebung entwickelt. Anhand von Laborexperimenten mithilfe des Prototyps konnten verschiedene theoretische Testszenarien in der Praxis getestet werden.

Die Resultate dieser Arbeit zeigen, dass HSMs geeignet sind, den physischen Schutz von VPNs zu verbessern. Obwohl sich dadurch Leistungseinbußen im Vergleich zu einem VPN ohne besonderen physischen Schutz ergeben, war der getestete Prototyp in der Lage, mehrere tausend VPN-Clients ohne Störung zu bedienen. Dies zeigt, dass HSMs auch in Hochleistungsumgebungen performant genug sind, um ein VPN abzusichern.

Keywords: *Hardware-Sicherheitsmodul, Virtuelles Privates Netzwerk, Leistungsmessung*

Abstract

A Virtual Private Network (VPN) enables encrypted access to private networks. In environments with particularly high security requirements, it may be necessary to protect the secret keys required for VPN operation from physical access to an exceptional degree. By storing the secret keys in a Hardware Security Module (HSM), these keys can be protected from physical attacks, but all cryptographic operations using these keys must take place within the HSM. Depending on the deployment environment, this may mean that the HSM has to perform several hundred to several thousand calculations per second. This thesis examines the suitability of HSMs for physically securing VPNs in high-performance environments.

As part of this thesis, a prototype of a performance testing environment was developed. Through laboratory experiments using the prototype, various theoretical test scenarios were tested in practice.

The results of this thesis show that HSMs are suitable for improving the physical protection of VPNs. Although this leads to performance losses compared to a VPN without any special physical protection, the tested prototype was able to handle several thousand VPN clients without interruption. This demonstrates that HSMs are powerful enough to secure a VPN even in high-performance environments.

Keywords: *Hardware Security Module, Virtual Private Network, Performance Measurement*

Contents

CHAPTER 1

Introduction

This chapter will present a detailed problem statement, the motivation, the expected results, the structure of this thesis, and the used methodology.

1.1 Problem Statement

A Hardware Security Module (HSM) is a hardened physical computing device that specialises in cryptographic key creation, key storage, and execution of cryptographic processes, as stated by Kamaraju, Ali, and Deepak [51]. HSMs come in different form factors, such as smartcards, PCI plugin cards, or Local Area Network (LAN)-based appliances, according to Ivarsson and Nilsson [48]. HSMs are used in a wide variety of industries, including banking, insurance, and blockchain applications, as stated by Sommerhalder [86]. HSMs can support a wide range of security operations, such as the generation and protection of Personal Identification Numbers (PINs), protecting transactions on the Internet, or automatic signature confirmation, according to Truong and Dang [93]. Another possible area of operations for an HSM is the key management for secure remote access.

A LAN is a network of systems within a small geographic area such as an office or building and is considered a private network, as stated by Elahi and Cushman [32]. Access to such a private network is only possible by being on-site. By connecting a private network to the Internet, it is possible to access the network from remote. Companies and organisations that handle sensitive data need to protect access to their private networks.

Communication over the Internet is generally unencrypted. Anyone who is able to see the communication can usually read it, as stated by Jyothi and Reddy [50]. A Virtual Private Network (VPN) allows secure and authenticated remote access to a private network by tunnelling all communication between two endpoints through a cryptographically secured channel, as stated by van Oorschot [97]. This also adds the benefit of an additional layer of encryption and integrity.

In highly security-sensitive contexts, the requirements for VPNs can state that the generation and storage of all cryptographic keys must be done in a secure and tamper-resistant environment. An HSM can provide logical protection by isolating cryptographic processes, such as key generation, from other operations. By storing generated keys on the device, an HSM can also provide physical protection. Some HSMs can provide additional security by providing tamper-proofing features, as stated by Sommerhalder [86]. A VPN can be extended with an HSM such that key management tasks are carried out by the HSM. This allows the separation of a network from the Internet, enforcing authentication, and securing the generation and storage of cryptographic keys used during the process.

In high performance environments, such as in cloud environments, companies and organisations typically have to satisfy some availability metrics. A VPN might be required to serve hundreds or even thousands of users. While commonly used VPN protocols have proven themselves in practice to be able to handle thousands of users, the suitability of HSMs in such a scenario is an open question.

1.2 Motivation

As stated before, HSMs are used in various industries, such as banking and insurance, and in critical infrastructure such as payment solutions and encryption schemes [86]. Although HSMs can be used to increase the security of a system, they can incur additional financial costs as well as cause a performance overhead (see, for example, Aref and Ouda [7]). This overhead is especially critical in high-performance environments.

HSMs are typically used in environments where high security requirements are specified. Due to the high security requirements they have to fulfil, HSMs are expensive to purchase. Depending on the design and the computing power provided, HSMs can cost several thousand euros.

The actual performance provided by HSMs can only be roughly estimated before purchase, based on data sheets provided by the manufacturer. An accurate assessment of the performance provided can help keep procurement costs low.

The motivation of this thesis is to strengthen the security of information systems by evaluating the suitability of HSMs securing remote access in production systems with a high-performance demand.

1.3 Expected Results

The main hypothesis of this thesis is that HSMs are suitable in securing the remote access in high-performance environments. To this end, this thesis aims to answer the following questions:

RQ1. To what extent does secure access to remote services influence performance from an end user's perspective?

RQ2. How can secure remote access to central resources be scaled?

RQ3. Which characteristics are relevant to assess the suitability of secure cryptographic key storage in a high-performance environment?

The usage of specialised hardware for cryptographic operations could improve performance in comparison to processing on general server hardware. However, calling an HSM for cryptographic operations introduces some overhead compared to carrying out these operations directly on the VPN host. The result of the first research question is to determine the performance impact on a remote access that uses an HSM for cryptographic operations and whether an overall performance increase can be achieved.

Central resources, such as files, databases, or repositories might have to be available to a multitude of users. Even if a current solution is able to serve the current number of users satisfactorily, it is necessary to take into account the needs of a growing user base. For this reason, secure remote access solutions must offer a way to scale easily and efficiently.

The third question aims to identify the characteristics that are relevant to determining the suitability of a cryptographic key storage for securing remote access in high-performance environments. Identifying the relevant characteristics allows for a specific selection of hardware or software to be made.

The expected result of this thesis is a prototype for testing secure remote access solutions. Additionally, this thesis will present a concept for scaling secure remote access solutions in high-performance environments.

1.4 Structure

Chapter 2 will present scientific research related to this thesis.

Chapter 3 will start by laying the groundwork for this thesis. It will first cover the fundamental aspects of Information Security that are relevant for VPNs. Following that, it will present the fundamental aspects of computer networks necessary for the rest of the thesis. Additionally, a brief discussion of the basic elements of cryptography required for this thesis will be provided. The chapter will also discuss general aspects of software testing. Lastly, a brief discussion on statistics will conclude the chapter.

Chapter 4 will discuss what an HSM is and how it is commonly used in practice. Two key aspects of HSMs, key management and physical security, will be discussed in more detail. Existing scaling methods for HSMs are discussed. The chapter concludes by presenting examples of actual HSMs to provide context for the thesis.

Chapter 5 will discuss VPNs in detail. Based on the foundations in Chapter 3, it will discuss how a VPN works and what its limitations are and will provide use cases for VPNs. A specific example of a VPN protocol will be presented at the end of the chapter.

Chapter 6 will discuss a specific type of software test, namely performance testing. It will provide an outline of the definitions, goals, purposes, and challenges of performance testing. Common performance testing types and strategies as well as performance metrics commonly captured during testing will be presented as well. The chapter will present how workloads are generated during performance testing. Finally, a general concept of a performance test environment is presented.

Chapter 7 will present the results of this thesis. It will start by discussing how the performance tests were carried out and which performance metrics were used. Afterwards, it will discuss how performance metrics were captured during testing and how the raw data was processed for visualisation. Finally, the chapter will present and discuss the performance test results in detail.

Chapter 8 will discuss the results of the thesis and Chapter 9 will summarise the contents of this thesis and provide an outlook on future work.

1.5 Methodology

In the beginning, literature research is carried out to determine the foundations and the state-of-the-art in the area of remote access, secure cryptographic storages, and performance testing.

A prototype is created to answer question RQ1. Performance test methods suitable for remote access testing are identified in the literature. This includes test types, inputs, and performance metrics. A prototype of a test environment for testing secure remote access solutions is devised. Performance tests of secure remote access solutions are carried out using the prototype. The first question is answered using the results of the performance tests.

For the purpose of answering RQ2, existing scaling methods are identified based on literature research. Suitable scaling methods are identified and evaluated in the course of a laboratory experiment within the test environment. For this purpose, the prototype is adapted or expanded if necessary. Performance tests are carried out and compared with previous results to determine whether the identified scaling methods are effective.

Based on literature research, common characteristics of secure cryptographic storages are collected. Afterwards, using the results of the prototype created for question RQ1 and the results of the laboratory experiment for question RQ2, the relevant characteristics can be identified or even extended.

CHAPTER 2

Related Work

Aref and Ouda [7] argue that security and privacy are paramount concerns for businesses and organisations and that mechanisms to protect integrity, confidentiality, and authenticity are necessary to allow the exchange of sensitive information across a network. Their work focuses on securing intra-domain communication, that is communication confined within a network administered by a single organisation. They argue that, in order to achieve secure intra-domain communication, Secure Socket Layer (SSL) needs to be integrated with all relevant networking protocols, which requires the use of numerous cryptographic keys and effective certificate management. HSMs can offer such security mechanisms. However, HSMs require the use of vendor-specific libraries for operation, making it hard to change vendors. The provided solution, named “HSM4SSL”, addresses this vendor issue by proposing a specific architecture where HSMs are provided as a service to multiple organisations. They claim that their solution can be easily expanded for more client organisations without compromising security. Performance measurements were carried out for up to 100 concurrent clients. The latency results show a linear increase of latency suggesting it is not able to handle thousands of clients.

Han et al. [42] propose a scalable and secure system for key management in a cloud environment. The proposed solution achieves multi-tenancy by providing multiple virtual HSMs within a single HSM device. Their solution incurs a performance degradation in the case of keyless SSL for Transport Layer Security (TLS) private key offloading compared to a stand-alone HSM. Performance tests were carried out for up to 32 concurrent clients.

Urien [96] argues that the off-loading of cryptographic operations into an HSM can increase the performance and security of a system. The author presents a solution for off-loading SSL/TLS operations and measures the performance achieved by the new approach. The results show a speed-up for specific SSL/TLS configurations.

Goethals et al. [38] argue that the edge devices used in Internet of Things (IoT) have become powerful enough to run virtual containers. VPNs can aid in the deployment and operation of containers on edge devices. They go on to argue that a suitable VPN must be able to handle a large number of simultaneous connections. They aim to evaluate the ability of VPN protocols to scale with the size of an IoT network. The VPN protocols considered in this paper were OpenVPN, WireGuard, ZeroTier, Tine, and SoftEther. They concluded that WireGuard showed remarkable performance and scaled better than the other protocols.

Mackey et al. [61] argue that a fundamental problem incurred by VPNs is the overhead they cause on the network throughput. They created an automated test framework to carry out a performance comparison between WireGuard and OpenVPN in a local environment and on AWS by using iperf3 as a load generation tool. Their results reveal that WireGuard outperforms OpenVPN on all testing setups and especially on multicore machines due to its efficient multi-threading code.

Chua and Ng [23] argue that there is a growing trend towards open source VPN solution. They carry out performance tests for the three popular open source VPN protocols WireGuard, OpenVPN, and OpenConnect across different client devices and deployment scenarios. The test setup includes different client devices and multiple network topologies. The tests focused on the bandwidth and speed of the VPN clients. They concluded that WireGuard performs best on platforms where the kernel version can be used and that performance on more restrictive systems where the kernel version cannot be used is an open question.

Antoniuk and Plechawska-Wójcik [5] argue that VPNs are increasingly being used by organisations as well as private individuals. Their thesis aims to compare the performance of WireGuard with that of older VPN protocols such as OpenVPN and Layer 2 Tunneling Protocol (L2TP). The performance tests were carried out using the *ping* tool, Speedtest-cli, and iperf3. Their results show that the performance of WireGuard decreases for a mobile client.

Akter et al. [2] argue that it is a critical challenge for organisations to select an appropriate VPN protocol in the plethora of protocols. The purpose of this paper is to determine the suitability of IPSec, L2TP, and Multiprotocol Label Switching (MPLS) for different service requirements. The performance tests focus on metrics such as throughput, Round-Trip Time (RTT), and jitter. They concluded that MPLS performs best in both elastic and time-sensitive applications.

The aim of Gentile et al. [36] is to build an Open Source infrastructure for Smart Devices in an IoT environment. They argue that VPNs can be used to create secure connections between user clients and remote endpoints but, the choice of the VPN protocol affects performance. The goals of this paper were to find the VPN protocol with the best efficiency, the widest spectrum of compatibility, and to find an open firmware on a constrained system that is compatible with different VPN protocols. The `iperf` tool was used to generate the workload for the performance tests and the *atop* tool was used to monitor the test targets. The result was an OpenWrt system that performed well with all tested VPN protocols.

Aung and Thein [9] argue that layer 2 VPN protocols allow the usage of software developed for LAN such as printer sharing, some database protocols and others. In their work, they compare different layer 2 VPN protocols such as L2TP, Point-to-Point Tunneling Protocol (PPTP), OpenVPN, Ethernet over Internet Protocol (EoIP), and MPLS. The Wide Area Network Emulator (WANem) was used to emulate an Internet connection and the `iperf` tool was used to generate throughput and measure performance. Protocols are compared with each other on the basis of security, scalability, and cost. The results of the paper are inconclusive and can not recommend a specific protocol over another.

Budiyanto and Gunawan [20] analyse the quality of service of Voice over Internet Protocol (VoIP) using different VPN protocols. Performance metrics considered were delay, jitter, throughput, and packet loss.

Ghanem et al. [37] argue that smart grid networks need secure and reliable communication. While wireless communication is effective to reach a large amount of devices spread over a wide geographic area it is necessary to secure this over-the-air communication. VPNs can be used to secure such networks. The paper aims to determine the overhead caused by utilising a VPN. Considered VPN protocols were OpenVPN and Internet Protocol Security (IPsec). The result of the paper shows that IPsec created a larger overhead compared to OpenVPN.

Nawej and Du [71] argue that VPN protocols can be an issue to network performance. Measurements were carried out in a simulated environment. No specific VPN protocol was examined, but rather a generic model of a VPN. Considered network performance metrics were throughput and delay. The result of the paper shows that VPNs have a negative impact on network performance.

None of the research described above considers the performance impact of extending a VPN with an HSM for the purpose of key management.

Donenfeld [29] presents the WireGuard VPN protocol in detail. The paper provides insight into the inner workings, stating the design principles and the cryptographic schemes and functions used. The paper concludes with a short section comparing the performance of WireGuard with IPsec and OpenVPN in a single-user setting.

In a recent paper by Master and Garman [62] the security design principles of the WireGuard protocol are summarised, and the protocol itself is reviewed and also implemented.

CHAPTER 3

Foundations

This chapter will lay the groundwork for all sections thereafter by presenting important concepts and definitions on the topic of information security in Section 3.1, computer networks in Section 3.2, cryptography in Section 3.3, software testing in Section 3.4 and statistics in Section 3.5.

3.1 Information Security

To understand the importance of VPNs and why measuring its performance is of interest, it is necessary to first understand the importance of information security.

3.1.1 Motivation and Goals

According to Sharp [85], "Information security is an old discipline, whose aim is to ensure that only the right people can get hold of and possibly also change or delete information". Although technology evolved, the goal of information security did not. In the early days of computer technology, during the mid-1980s, when mainframe and mid-level computers dominated the market, according to Bishop [18], computer security problems and solutions centred on securing files on single systems. Sharp [85] points out that information security was only relevant to governments and the military especially since they were the first users of computers. However, with the rise of the Internet, computer security changed focus to the new network environment, according to Bishop [18].

Sharp [85] agrees by stating that in the early days of computer technology, the "focus was mostly on large centralised computers, where user terminals, if any, were directly connected to the computer by dedicated cables rather than a network as we know it today." Additionally, when the Internet was first developed it was mainly used by a small group of technically proficient people who were willing to accept a certain risk in making use of it. In short, security was not an important issue. This changed when more and more services were developed and made available on the Internet, such as banking, commerce, or public administration, causing people without technical backgrounds to begin to use the Internet without understanding the risks of using it.

Schneier [82] points out that "Although attacks in the digital world might have the same goals [. . .] as attacks in the physical world, [. . .] They will be more common" because digital attacks can be automated. Kappes [52] argues that know-how has become the most important economic factor today for defining the value of a company and that ever more information, which makes up the know-how of a company, resides in digital form which is why physical security measures alone are insufficient.

The three main goals commonly defined for information security are *confidentiality*, *integrity*, and *availability* (see, for example, [85, 52]) which are commonly referred to as the *CIA triad*.

- **Confidentiality:** Van Oorschot [97] states that the aim of this goal is that non-public information, stored and in transit, is only accessible to authorised parties. Bishop [18] adds to this an important aspect by arguing that the mere existence of data can also be confidential.
- **Integrity:** Property of data, software, and hardware is only alterable by authorised parties [97]. National Institute of Standards and Technology (NIST) is using a more comprehensive definition by which this goal aims at "Guarding against improper information modification or destruction, and includes ensuring information non-repudiation and authenticity" [94]. Sharp [85] makes an important distinction that "assets can only be modified by authorised parties in authorised ways".
- **Availability:** Van Oorschot [97] states that the aim of this goal is that information, services, and computing resources remain accessible for authorised use. An attacker could target this goal by deliberately denying authorised users access to systems or services [18].

Some authors add additional security goals to the previously discussed one, such as *authenticity* and *non-repudiation* [52].

- **Authenticity:** Kappes [52] defines this goal as the unambiguous identification of the sender during communication and as the unambiguous identification of a communication party, in general. Alsmadi et al. [3] explain that during authentication a user proves his claimed identity. NIST [68] is using a more detailed definition by enumerating "user, process, or device" as relevant parties for which the identity must be verified. In addition, identity verification might often be a prerequisite before access to resources is allowed.
- **Non-repudiation:** Bishop [18] argues that repudiation of origin is a false denial that an entity sent something. Kappes [52] states that this security goal aims to

allow the sender and contents of a message to be assigned to a user. Alsmadi et al. [3] expand this from the sender not being able to deny sending a message to the receiver also not being able to deny receiving a message. A more general definition is given by van Oorschot [97] who describes the aim as "the ability to identify principals responsible for past actions".

Section 5 will discuss how VPNs can help protect the digital information of a company by supporting some of the information security goals discussed in this section, such as confidentiality, integrity, and authenticity.

3.1.2 Threat Sources and Motives

To understand how a VPN can aid in achieving security goals it is first necessary to understand where threats to digital information stem from and what motivation lies behind them.

Bishop [18] defines a *threat* to be a potential security violation. According to the NIST [22], a *threat* is "the potential cause of an unwanted incident, which can result in harm to a system or organization". Eckert [31] defines a *threat* as the intention to use one or more vulnerabilities to undermine one or more security goals. All definitions for *threat* have in common that they describe a potential event that harms security goals.

Threats can arise from different sources. Sharp [85] distinguishes four groups of threats:

1. *Hardware related threats:* physically affect the system, such as heat, water, or dust.
2. *Software related threats:* affect the software on the system and can be attributed to flawed software or intentionally malicious programmes.
3. *Data related threats:* can lead to unauthorised processing of data, such as modification, disclosure, or deletion.
4. *Liveware related threats:* stem from human errors by system users and cover mistakes as well as intentional attacks.

Intentionally malicious programmes and intentional attacks are created and carried out by attackers. These attackers can have different motives for their actions. Lincke [59] distinguishes three different motives.

- *Cybercrime* relates to all attacks that have criminal intent.
- *Espionage* is concerned with the theft of intellectual property.
- *Information Warfare* is the act of attacking critical infrastructure.

Schneier [82] provides a more nuanced discussion and groups adversaries, who are threatening the digital world, into several groups:

- *Hackers* describe persons "with a particular set of skills and not a particular set of morals". Hackers are interested in the inner workings of a system to such an extent that they can have more expertise about a system than the designers of the systems themselves.
- *Lone Criminals* usually don't have much expertise and can be insiders or outsiders who notice a flaw in a system and decide to exploit it. They are responsible for most computer crimes.
- *Malicious Insider* is already inside the system and may be regarded as trustworthy with a high level of access to sensitive systems. The malicious insider could be motivated by revenge, financial gain, or publicity.
- *Industrial Espionage* aims at illegally acquiring competitor trade secrets to gain an advantage over competition.
- *The Press* is interested in getting a newsworthy story and can be motivated by financial gain or by the belief in a just cause.
- *Organized Crime* pursues financial gain and can purchase expertise to attack financial systems.
- *Police* is interested in collecting information and is not above breaking the law.
- *Terrorists* are typically motivated by geopolitics, moral or ethical beliefs, and rather interested in causing harm than anything else.
- *National Intelligence Agencies* are capable and dedicated adversaries interested in gaining access to military information, weapons designs, and diplomatic information but can also be linked to industrial espionage.
- *Infowarriors* are military adversaries that focus on the short-term goal of affecting the ability of their target to wage war.

Knowing about the different sources of threats and the motives and intentions of attackers can help to assess the risks a system faces, which will be discussed in the next section. A VPN can help protect individuals and organisations from malicious actors.

3.1.3 Risk Assessment and Management

Assessment of risks is an important first step in identifying threats to information security goals. According to Eckert [31], the importance of a threat depends on the context. For example, unauthorised read access does not pose a threat to a public database, but it certainly does to a private database. A risk assessment can help to determine which threats are relevant for a given context. Schneier [82] argues that it is necessary to understand the real threats to the system and assess the risk of these threats. Otherwise, the system might not be protected against the threats that matter.

A wide array of different definitions of risk can be found in the literature. Eckert [31] defines the *risk* of a threat as the probability of the occurrence of a damaging event and the amount of potential damage that can be caused by it. Parkinson [74] defines *risk* as the quantitative probability that an error situation occurs. Kappes [52] states that *risk*, in the context of computer systems, is the probability that an existing vulnerability is being used for an attack. According to the definition by the NIST [40] a *risk* is

> [. . .] a measure of the extent to which an entity is threatened by a potential circumstance or event, and is typically a function of: (i) the adverse impacts that would arise if the circumstance or event occurs; and (ii) the likelihood of occurrence. Information security risks are those risks that arise from the loss of confidentiality, integrity, or availability of information or information systems and reflect the potential adverse impacts on organisational operations.

Crouhy, Galai, and Mark [26] specifically define *cybersecurity risk* as "vulnerabilities of information or information systems and any related consequences". Rausand and Haugen [78] point out that the word "risk" has varying meanings in the general public and the scientific community, ranging from "change", "likelihood", and "possibility" to "hazard", "threat", or "danger". Rausand and Haugen [78] go on to define *risk* as

> The combined answer to the three questions: (1) What can go wrong? (2) What is the likelihood of that happening? and (3) What are the consequences?

Bishop [18] argues that potential threats to a system and the likelihood that they occur must be assessed to determine whether an asset should be protected and to what extent. According to Schneier [82], risk assessment considers all threats and estimates the expected loss per incident and the expected number of incidents per year. A definition provided by the American National Standards Institute [76] defines *risk assessment* as "a process that commences with hazard identification and analysis, through which the probable severity of harm or damage is established, followed by an estimate of the probability of the incident or exposure occurring, and concluding with a statement of risk". Rausand and Haugen [78] differentiate between *risk analysis* as "systematic study to identify and describe what can go wrong and what the causes, the likelihoods, and the consequences might be" and *risk assessment* which is "The process of planning, preparing, performing, and reporting a risk analysis, and evaluating the results against risk acceptance criteria".

Once risks have been identified, they have to be addressed, constantly monitored, and periodically re-assessed. This process of addressing, monitoring, and re-assessing is referred to as risk management. Sharp [85] states that *risk management* "deals with all the activities which are related to evaluating and reducing risks". Lincke [59] similarly states that *risk management* “includes all stages of managing risk” and goes on to define the stages as:

1. Establish scope and boundaries
2. Risk assessment
3. Risk treatment

Rausand and Haugen [78] define the *risk management* process similar to Lincke [59] but point out that risk management is an iterative process.

An important distinction between *risk* and *uncertainty* has been made by Knight [54] in 1921 where risk is a variability that can be quantified in terms of probabilities whereas uncertainty is a variability that cannot be quantified at all. Schneier [82] argues that “there’s going to be a lot of guesswork” in the risk assessment because “the particular risks we’re talking about are just too new and too poorly understood to be better quantized”.

Kappes [52] also argues that neither the probability of the occurrence of a damaging event nor the amount of potential damage can be quantified in practice. Kappes [52] goes on to say that not all risks can be completely mitigated but that there remains some residual risk.

Rausand and Haugen [78] state that “All results from a quantitative risk analysis are uncertain to some degree”. This *uncertainty* can have different causes, ranging from the use of inadequate models to the failure to identify potential threats.

Determining threats and assessing risks is not a precise science, but rather relies on expertise. In addition, new threats can emerge at any time that were unknown at the time of the risk analysis. These are some reasons why complete security can not be achieved.

3.2 Computer Networks

As Sharp [85] points out, “computers seldom work completely alone”, many tasks which computers solve require other computers. Similarly, Tanenbaum and Wetherall [91] argue that in the past an organisation’s computational needs could be met by a single computer, but over time needs changed such that numerous interconnected computers, called *computer networks*, are necessary to meet them. This section will discuss the design and organisation of computer networks.

3.2.1 Types of Networks

According to Sharp [85], computer networks consist of nodes, connected wirelessly or by cable. Users connect their end systems to these networks as hosts. Computer networks can be classified by the size of the area the network should cover.

Personal Area Network (PAN): A PAN covers communication “over the range of a person”, according to Tanenbaum and Wetherall [91], or “just a few meters” [85]. An example of a PAN would be the connection of peripherals, such as a mouse or keyboard, to a computer using Bluetooth [91, 31].

Local Area Network (LAN): A LAN spans across “an apartment, a building, company site or a university campus”, according to Baun [12], or in other words, it spans “a few kilometers” [85]. They are privately owned networks, used to connect personal computers such that they can exchange information and share resources, such as printers [91].

Metropolitan Area Network (MAN): Different sizes for a MAN can be found in literature, such as covering a city [91], “extending from 2 to 50 km” [80] or extending across a big city with a maximum expansion of up to 100 km [12]. A MAN connects several LANs and can be considered the middle ground between a LAN and Wide Area Network (WAN) [80]. Early examples of MANs are cable television networks [91].

Wide Area Network (WAN): Most sources state that a WAN covers a country or continent [91, 12, 80, 85]. Some sources extend the range even to the entire world [80, 85].

Tanenbaum and Wetherall [91] discuss WANs in great detail, stating that they are used to connecting offices that contain hosts to each other. The part of the WAN connecting the hosts is called the *subnet*, consisting of transmission lines and switching elements. It should be noted that the term *subnet* was used to only refer to this part of the network but has gained additional meaning in the context of network addressing. Two important distinctions are made between LANs and WANs. The first difference is that the hosts and the subnet are owned and operated by different people. The second difference lies in what is connected to the network. While only individual computers are connected to a LAN, in the case of a WAN it could be individual computers or even entire LANs, or even MANs [12].

Global Area Network (GAN): Baun [12] introduces the GAN as a WAN that is covering an unlimited geographic area. Examples of GANs are the Internet or globally distributed offices of a company.

3.2.2 Networking Models

Tanenbaum and Wetherall [91] explain that networks are organised in *layers* or levels. The number and purpose of each layer can differ from network to network. A *protocol* defines how communication between layers is organised and will be discussed further in Section 3.2.4. The combined set of layers and protocols is called a *network architecture*. There are two important network architectures which are typically discussed in the literature (see, for example, [91, 52, 80, 12]):

1. The Open Systems Interconnection (OSI) Reference Model

2. The TCP/IP Reference Model

The OSI Reference Model: According to Kizza [53], in the early days of computing, computers were standalone machines where the moving of files between computers was done manually by removable disks. Due to the need for multiple computers to be able to talk to each other, the International Standards Organization (ISO) was tasked with developing the OSI model. The model consists of seven layers. While the logical data flow occurs between layers of the same level, the real data flow is more complicated. A message sent from system A to system B is passed down through the layers, each overlying layer passing the message on to the next underlying layer until the message is finally sent over a physical medium to the next system by the lowest layer. Each layer adds control information that is used during communication. A message received at system B takes the opposite direction through the layers where each layer is using the provided control information to process the message. For higher layers, data transfer appears as an end-to-end communication. Figure 3.1 presents the logical and real flow of messages in this model.

Following is a description of the seven layers of the OSI model [80]:

1. **The Physical Layer** is concerned with the transmission of raw bits over the network. Its main focus lies on the mechanical, electrical, and timing issues of transmission. It has to define how 1-bit and 0-bit are represented.

2. **The Data Link Layer** packs the packets of the network layer into *frames*. It attaches a checksum to each frame, allowing it to identify randomly occurring transmission errors.

3. **The Network Layer** has the main task of finding a suitable path for the messages to be transmitted by *routing* them from source to destination.

4. **The Transport Layer** enables the overlying layers to establish end-to-end communication channels between two systems.

5. **The Session Layer** establishes, manages, and terminates connections or *sessions* between applications on different machines.

6. **The Presentation Layer** is concerned with the syntax of a message to allow computers with different internal data representations to communicate with each other.

7. **The Application Layer** provides a variety of protocols that provide services to users, such as Hypertext Transfer Protocol (HTTP).

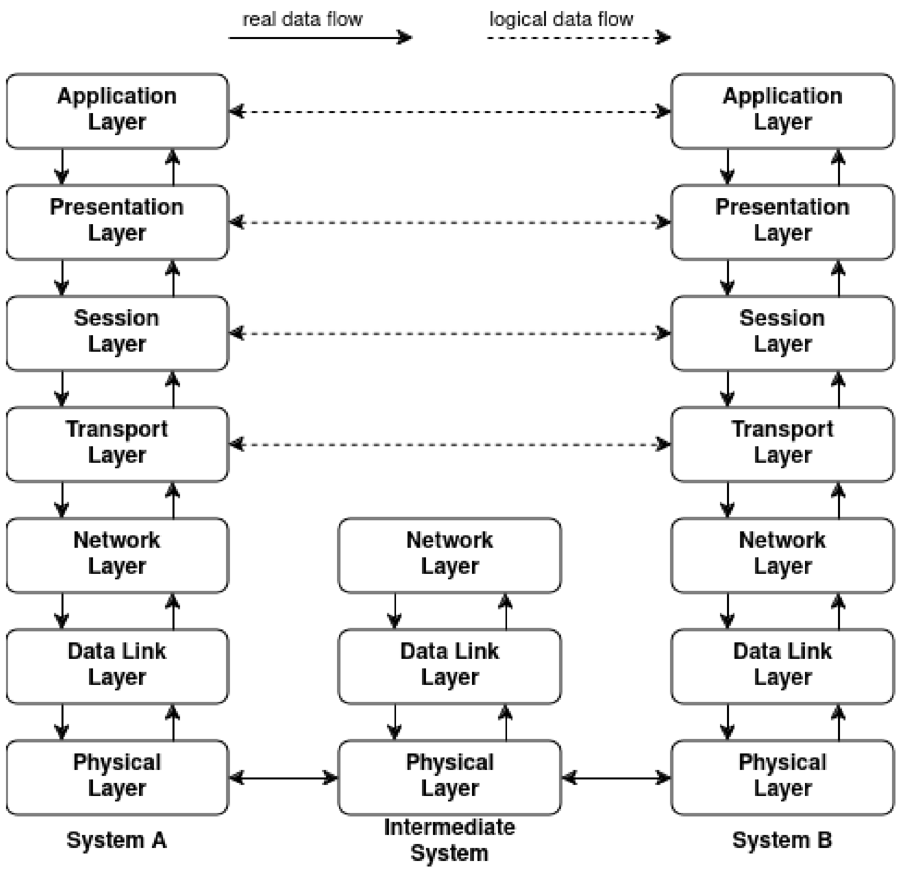


Figure 3.1: Layers of the OSI Reference Model [31].

The TCP/IP Reference Model: According to Tanenbaum and Wetherall [91], the TCP/IP Reference Model was used in the Advanced Research Projects Agency Network (ARPANET) and the worldwide Internet. The ARPANET was a research network that eventually connected hundreds of universities and government installations. The name of the TCP/IP model comes from the two primary protocols used in this architecture. Major design goals were the ability to connect multiple networks seamlessly even if some machines between source and destination would suddenly be put out of operation.

The model is organised into four layers [12] or five layers [80].

- **The Link Layer**: is responsible for the transmission of raw bits over various media. Can be split into *physical layer* and *network interface layer* in a five-layered model.
- **The Internet Layer**: enables hosts of a network to insert messages to the network and route them to their target independently. The messages could arrive out-of-order at the destination, which requires that higher layers might need to rearrange them.

- **The Transport Layer**: is concerned with host-to-host communication and is responsible for the reliability, flow control, and correction of messages.

- **The Application Layer**: contains all higher-level protocols such as HTTP, Secure Shell (SSH), File Transfer Protocol (FTP), or Simple Mail Transfer Protocol (SMTP).

3.2.3 Network Devices

A computer network is a system of loosely coupled devices that can communicate with each other [53]. The end systems of a computer network are commonly called *hosts*, and the intermediate switching devices are called *network elements* [53]. Network elements can be attributed to the type of network they are used in or to the layer of the network architecture they operate on. This section will present network devices relevant to this thesis.

- **Router:** Routers operate mainly on the network layer [80, 53]. They are used to interconnecting two or more networks [53] such as connecting a LAN to a WAN [80, 12]. Routers maintain dynamically updating routing tables [80, 53] which they use to determine the best, fastest, and most efficient way of routing a packet to its destination [53].

- **Gateway:** Gateways, which are also known as protocol converters [80] or protocol translators [12], perform all functions of a Router and more [53] and operate at any network layer [80]. They can perform protocol conversion between different types of networks, architectures, or applications [53, 12]. They can be divided into *transport gateways* that connect computers using different transport protocols and *application gateways* that can translate messages between different formats.

3.2.4 Network Protocols

According to Baun [12], *protocols* are an agreement between participants of a communication inside a computer network on how to structure their messages. As discussed in Section 3.2.2, network architectures are separated into layers. Each layer of a network architecture has its protocols, defining the syntax and semantics of messages exchanged, starting at the transmission of bits on the bottom layer to specifying how information is displayed at the higher layers. Each layer addresses certain aspects of the overall communication and provides an interface to the layer above and below. An interface provides a set of operations that put together define a *service*.

Tanenbaum and Wetherall [91] argue that a *service* provides its interface only to the layer above it, where the lower layer is the service provider and the upper layer is the service user. The service defines operations it is ready to perform without revealing how the operations are implemented.

Two network protocols are important to this thesis, the Internet Protocol (IP) and User Datagram Protocol (UDP).

IP: The IP forms the backbone of today's Internet [52, 91]. It is located on the network layer [83, 91]. Its first version *IPv4* was designed in 1981 at a time when it was impossible to predict how the Internet would develop [52]. Its successor *IPv6* was supposed to address some shortcomings such as the small address space [52]. It is a connection-less protocol, meaning packets can be sent without having to establish a connection to the destination prior [83]. Each packet could get routed over different paths, without the sender knowing whether packets were received by intermediate nodes or the receiver [52]. Packets are being transported on a best-effort basis, meaning that the receiving of the packets at the destination is not guaranteed [91].

IP is split into *header* and *body payload* [91]. The header consists of a fixed part of 20 bytes and a variable length optional part [91] and is shown in Figure 3.2. Schwenk [83] describes the parts of the IP header as follows:

- **Version:** Contains the version this IP packet is using. Allowed values are 4 and 6.
- **IHL:** The Internet Header Length (IHL) describes the length of the header in multiples of 32 bits.
- **Differentiated services:** This field allows routers to assign preference to IP packets.
- **Total length:** This field contains the total length of the entire IP packet in bytes.
- **Identification, Flags, and Fragment offset:** Each network limits the packets travelling on it to a maximum size. By splitting up larger packets into fragments, they can be sent over the network [91]. These fields are used to control the fragmentation of the IP packet.
- **Time-to-live:** This field's value is decremented by each router it passes through, preventing IP packets from circulating endlessly.
- **Protocol:** This field specifies which transport layer protocol the packet is transporting.
- **Header checksum:** This checksum is calculated over all header fields and must be recalculated if some header fields change, such as the time-to-live field.
- **Source address:** The source address of the IP packet, allows returning responses or error messages to the sender.
- **Destination address:** The IP packet is forwarded to this address on the network.

Version 4 bits	IHL 4 bits	Differentiated Services 1 byte	Total length 2 bytes
Identification 2 bytes			Flags and Fragmentation offset 2 bytes
Time to live 1 byte		Protocol 1 byte	Header checksum 2 bytes
Source address 4 bytes			
Destination address 4 bytes			

Figure 3.2: The IPv4 header [91].

Source port 2 bytes	Destination port 2 bytes
UDP length 2 bytes	UDP checksum 2 bytes

Figure 3.3: The UDP header [91].

UDP: According to Kappes [52], UDP provides a *connection-less*, *best-effort service*. A *connection-less* service does not have to establish a connection with the receiving system. An application passes the UDP service a *datagram* which the service can send out immediately. The term *best-effort* refers to the fact that UDP will not try to resend lost datagrams. A UDP message consists of a header and a payload [91]. The UDP header consists only of four parts [80], as shown in Figure 3.3:

- **Source port:** Contains the address of the application that created the message.
- **Destination port:** Contains the address of the application that receives the message.
- **UDP length:** Contains the total number of bytes of the message, including the header.
- **UDP checksum:** This optional field is used for error detection [91].

3.2.5 Static and Dynamic Routing

Network devices, such as routers, need to know where to route received data packets. According to Sadiku and Akujuobi [80], there are two ways through which a router can receive this information:

1. **Static Routing:** In this case, the routing information is stored in the routing table manually [80]. Because routing decisions are made in advance, it is not possible to

respond to failures [91]. Since the routing table has to be updated manually, static routing is only feasible for small environments [80].

2. **Dynamic Routing:** Dynamic routing algorithms change their routing decisions based on changes in the network or network traffic itself [91]. Routers can acquire necessary information by using certain protocols [80] such as the Intermediate System to Intermediate System Protocol (IS-IS) or Open Shortest Path First (OSPF) protocol.

3.2.6 Computer Network Performance

This thesis will measure the performance of the WireGuard VPN protocol. This section will take a look at the performance metrics of computer networks in general. Section 7.2 will then go on and discuss performance metrics for testing WireGuard in particular.

Obaidat and Boudriga [73] provide common goals for the evaluation of the performance of communication networks. Of interest for this thesis are the following:

- **Capacity planning:** The goal of capacity planning is to make sure that sufficient resources are available to meet the demand cost-effectively. Steps during capacity planning consist of instrumenting the system, observing the system, selecting the workload, and selecting the best configuration.
- **Performance debugging:** In case a system is working as intended but displays a poorer performance than expected it might be necessary to carry out a performance analysis.

Three general methods for evaluating the performance of a computer and telecommunication system can be discerned [73]:

1. **Analytic Modeling:** Analytic models provide an approximation of the test system and are computationally inexpensive.
2. **Simulation:** Simulations provide flexible, accurate, and credible results. However, designing, verifying, and validating the model requires a considerable time investment.
3. **Measurement and Testing:** While real measurements and experiments on a test system or the actual system are the most expensive, they provide the most accurate results.

An important part when evaluating the performance of a system is determining adequate performance metrics. Lilja [58] states that a *performance metric* is a measured value that describes the performance of a system. According to Obaidat and Boudriga [73], performance metrics should be relevant and meaningful and allow a comparison between systems. The basic characteristics that can be measured are [58]:

- *Count:* For example, how many requests were received, or how many requests were not answered.
- *Duration:* It is typically of interest how long some operation takes to return a value.
- *Size:* The size of a response or the storage a system requires could be of interest.

Sadiku and Musa [81] provide a list of common performance metrics for computer networks:

- **Capacity:** A counted metric that provides a value for the quantity of traffic with which a network can cope. This metric provides an upper limit which the network can handle in theory. It does not provide any insights into the actual performance of a network but is useful for planing networks.
- **Throughput:** This counted metric expresses how much traffic was received successfully at the intended destination. The throughput can be at most equivalent to the capacity of the network. Although it is a vital metric, it does not tell about data loss or whether the data arriving at the destination was corrupted. It is an important metric in high traffic environments.
- **Latency:** The latency is a duration-based metric that describes the time required to transmit a certain amount of data. It does indicate the responsiveness of a system. However, this metric is comparatively volatile, potentially increasing at times of high load. It is a crucial metric in time sensitive communication systems.
- **Loss Probability:** Traffic can get lost during communication. This counted metric expresses the probability of traffic being lost. A high loss probability can increase latency if data has to be re-sent. A low loss probability indicates a reliable network which can be crucial for ensuring the integrity of systems. However, low loss probability alone does not indicate a fast network. Other metrics such as throughput or latency could still be an issue. Thus, this metric is best used in conjunction with other performance metrics.
- **Queue length:** This measure provides the length of a buffer in case a queue is necessary during communication and is a size-based metric. A long queue can indicate congestion in the network or simply high traffic in normal operation. As such, this metric alone can be misleading and is best used for fine-tuning performance.
- **Jitter:** Jitter is the change in delay from packet to packet, or in other words, the variation in packet delivery timing. Similar to queue length, it can indicate network congestion. However, the root cause of high jitter can be manifold, and thus this metric is often used for fine-tuning performance. This metric is of importance for streaming services [73].

This thesis will focus on the two fundamental metrics in which network performance is measured: *throughput* and *latency* [12, 75]. Peterson and Davie [75] discuss the relationship between throughput and latency. While throughput and latency can define the performance characteristics of a network, their importance depends on the application. For a client sending only a 1-byte message and receiving a 1-byte response in return, the latency of the network will have more impact on the communication than the throughput. In contrast, a client trying to receive 25 MB will be more affected by the throughput the network can provide than the latency. Throughput, and implicitly capacity, is a central performance metric for a VPN server as it directly affects the number of users that can work in parallel. In the case of the WireGuard VPN regarded in this thesis, VPN clients are expected to send messages of a few hundred bytes as well as multiple megabytes. The latency of WireGuard handshakes, which will be discussed in more detail in Section 5.4.2, directly affects the performance of a WireGuard VPN as a handshake needs to be exchanged before encrypted data can be sent. Because of this, throughput as well as latency are considered to be of the most importance and will be discussed in more detail here.

Throughput: The terms *bandwidth*, *capacity*, *throughput*, and *data rate* can be found in the literature with slightly different meanings.

Peterson and Davie [75] explain that *bandwidth* used to denote the width of a frequency band. A telephone line supporting the frequency from 300Hertz (Hz) to 3300Hz would have a bandwidth of 3000Hz. In the context of computer networks, bandwidth typically refers to the number of bits per second a communication line can transmit. According to Peterson and Davie [75], this is also sometimes called the *data rate*. Peterson and Davie [75] go on to define *throughput* as the measured performance of a communication line whereas *bandwidth* denotes the maximum data rate. In contrast, Sadiku and Musa [81], as discussed above, define the maximum data rate as the *capacity* and uses the term *throughput* similar to Peterson and Davie [75]. This thesis will use the term *capacity* as denoting the maximum data rate of the communication line and *throughput* as denoting the measured data rate of the test target.

The throughput of a communication line is given by the number of bits that can be transmitted in a certain period [75]. The unit that is typically used is *bits per second (bs)* or sometimes *bytes per second (Bs)*. For example, a throughput of *1Mbs* states that the network can transmit one million bits in a second. Another way of thinking about throughput is that in a network of 1Mbs it takes 1 *microsecond (µs)* to transmit a single bit [75].

In order to provide context to the performance measurements in Section 7, this paragraph will provide some common network capacity values. *Ethernet* is one of the most pervasive network access methods used, commonly used in LANs [80]. Tanenbaum and Wetherall [91] provide a detailed description of different Ethernet standards. *Classical Ethernet* used a single long cable around a building where all computers were attached. These cables were limited by a maximum length of a couple of hundred metres and could only connect around 100 computers. Repeaters (see Section 3.2.3) had to be used to extend the networks. This early form of the Ethernet was standardised for a capacity of 10Mbs. Moving away from the single long cable architecture, a station called *hub* (see Section 3.2.3) was used where each computer was connected by a dedicated cable. However, the hub architecture was logically equivalent to the single cable architecture, thus not increasing the capacity of the network. The *switched Ethernet* was developed to address the increasing capacity demand by using switches instead of hubs (see Section 3.2.3). A switch was able to use the available capacity of 10Mbs more efficiently by only forwarding frames to destined ports. To meet the continually increasing capacity demand, further Ethernet standards were devised. *Fast Ethernet* increased the capacity of the network from 10Mbs to 100Mbs and *Gigabit Ethernet* up to 1000Mbs (or 1Gbs). The *10-Gigabit Ethernet*, with a capacity of 10Gbs, was intended to be used in data centres and with MANs. According to Gumaste [41], standards for *200-Gigabit* and *400-Gigabit* were ready in 2020 and the arrival of *terabit Ethernet* was just a matter of time. The highest currently available Ethernet capacity is standardised in IEEE P802.3df consisting of *800-Gigabit Ethernet* and was approved in 2024 [46].

Latency: The terms *latency* and *delay* are either used synonymous or with different meanings in the literature. To avoid confusion, throughout this thesis the term *latency* will denote the sum of all *delays* that can occur when a message is transmitted. Peterson and Davie [75] state that *latency* can be thought of as consisting of three parts:

1. **Propagation delay:** The transmission speed of any message can not surpass that of the speed of light. A lower bound for this delay can be calculated by dividing the distance the message has to travel over the speed of light. However, light travels across different media at different speeds, resulting generally in higher propagation delays.

2. **Transmit delay:** The transmission delay depends on the size of the packet that has to be sent and the capacity of the network. If, in theory, the packet size would only be one bit, the transmission delay becomes irrelevant [12].

3. **Queue delay:** Intermediate devices inside the network, such as switches and routers, might introduce some delay as these devices generally need to store packets for some time before being able to process and forward them.

The total latency can thus be defined as $Latency = Propagation\ delay + Transmit\ delay + Queue\ delay$. According to Peterson and Davie [75], rather than the one-way latency, the *RTT*, which indicates how long it takes to send one message from one end of the network to the other and back, is sometimes of more interest.

RIPE Atlas [79] is a global open measurement network, measuring Internet connectivity. It provides geographically distributed devices to carry out active measurements, as stated by Alvarez and Argyraki [4]. To present some typical RTT values for Austria, the open

Target: Austria (193.171.255.2)	
Austria	8.152 ms
Germany	16.554 ms
France	18.96 ms
Netherlands	17.983 ms
Hungary	4.515 ms
Sweden	29.883 ms
USA	93.726 ms

Table 3.1: Minimum RTTs from different countries to Austria.

database was searched for measurements targeting IP addresses in Austria. Some of the results are presented in Table 3.1.

3.2.7 Network Security and Threats

Schneier [82] argues that "network security goes hand in hand with computer security" since many devices are nowadays connected to the Internet. Since it is already difficult to secure a standalone computer, it is even more difficult to secure a connected computer because attackers can use the network to access the system.

Sharp [85] argues that wanted network traffic needs to be protected such that the appropriate information security goals, as discussed in Section 3.1.1, are met while preventing unwanted network traffic from passing. Sharp [85] goes on to argue that threats to meeting the security goals come from three fundamental difficulties:

1. **Physical Security:** It is difficult or even impossible to assert that all physical connections in a network are protected. Hence, it should be accepted that all traffic passing through a network could potentially be monitored, recorded, or modified. As a result, security goals discussed in Section 3.1.1 cannot be achieved without further measures.

2. **Limited Bandwidth:** One possible threat to the *availability* security goal is the fact that all communication networks have a limited capacity. By overloading a communication network, it becomes unavailable for other transmissions.

3. **Authentication:** Asserting the identity of a communication partner is difficult because it is not possible to directly see the other partner. Protocols such as IP or UDP do not provide any form of authentication (see Section 3.2.4).

Schwenk [83] is dividing threats encountered on the Internet into two categories, *passive* and *active* threats:

1. **Passive Threats:** Passive attacks aim at acquiring data transmitted over a network. This could break the *confidentiality* of the transmission. Using a private network can help protect against such a threat.

2. **Active Threats:** In an active attack, the attacker is modifying transmissions or even inserting fake transmissions. Modification of messages would break the *integrity* of transmissions while faking transmissions would break *authenticity*. The *availability* could be broken by dropping intercepted packets instead of passing them on.

Kappes [52] argues that the transmission between two participants could be disrupted in the following four ways:

1. **Intercept:** By intercepting a message an attacker could read the contents of the message.

2. **Manipulate:** An attacker could change transmitted messages.

3. **Deceiving:** An attacker could impersonate the sender, receiver, or even both ends of a communication.

4. **Interrupt:** An attacker could disturb the communication by interrupting the transmission.

Similar to the threats discussed by Schwenk [83], these four attacks target the *confidentiality*, *integrity*, *authenticity*, *availability*, and *non-repudiation* of the network.

3.2.8 Cloud Computing Model

According to Surianarayanan and Chelliah [90] different computing models fit different needs. *Cloud computing* is an Internet-based computing model where resources are rent out in order to efficiently utilize computing resources.

Two major cloud models can be distinguished, public and private cloud. According to Sehgal, Bhatt, and Acken [84] a public cloud offers its services to a wide range of users that can be anywhere around the world. In a private cloud access is restricted to a certain set of users and can be accessed over a private LAN.

An example of a cloud computing application, according to Surianarayanan and Chelliah [90], is the storage of patient data in health care contexts. Personal information and health data can be stored in the cloud and accessed by doctors anytime and anywhere.

Surianarayanan and Chelliah [90] discuss security threats related to cloud computing models. When data is being sent between user and cloud an attacker could passively intercept it. This would result in a loss of confidentiality (see Section 3.1.1). By using encryption this *eavesdropping* attack can be prevented. During a *man in the middle* attack, an attacker sitting between the user and the cloud could impersonate the user by actively intercepting and changing messages. Such an attack can be prevented by utilising authentication and message integrity checks. Section 3.3 will discuss the prevention methods in more detail.

3.3 Cryptography

Schwenk [83] argues that cryptographic methods are essential for protecting Internet communication and can be used to support the security goals discussed in Section 3.1.1. After introducing fundamental definitions in Section 3.3.1 the Sections 3.3.2 through 3.3.6 will discuss cryptographic primitives that are used by WireGuard. A detailed description of WireGuard is given in Section 5.4.

3.3.1 Motivation and Fundamentals

Literature provides different meanings of cryptography. Eckert [31] discusses that cryptography is an old teaching about the encryption and decryption of messages to keep information confidential from third parties. Similarly, Bishop [18] argues that the word cryptography comes from the Greek meaning 'secret writing' and that it is the science of concealing meaning with the aim of keeping enciphered information secret. Today, cryptography provides the cornerstone for secure communication. Sharp [85] discusses that cryptography is a fundamental technique to prevent the disclosure of confidential data and in its modern form it can be used to achieve security goals such as confidentiality, authenticity, and non-repudiation. Wong [100] explains that cryptography is "the science aiming to defend protocols against saboteurs". Menezes, Van Oorschot, and Vanstone [67] define cryptography to be "the study of mathematical techniques related to aspects of information security such as confidentiality, data integrity, entity authentication, and data origin authentication". Taken together it can be stated that cryptography is concerned with the protection of security goals by utilising mathematical techniques to allow for encryption and decryption of messages.

Cryptography considers three participants in a two-party communication, the *sender* which is the legitimate transmitter of information, commonly referred to as *Alice*, the *receiver* which is the intended recipient of information, commonly referred to as *Bob*, and the *adversary*, commonly referred to as *Eve*, which tries to defeat the security goals [67]. A *plaintext* m is the original, understandable form of a message and the *ciphertext* c is the scrambled form of the message [85]. *Encryption* is the method of disguising the actual data of a plaintext, *decryption* is the reverse process, as stated by Rao and Nayak [77]. An *encryption scheme* consists of an encryption function E and a decryption function D such that $D(E(m)) = m$ [100]. By using an encryption scheme, Alice and Bob can send messages over an unsecured channel without Eve being able to read the content of the messages. Figure 3.4 presents the fundamental notions of a cryptographic scheme, as described above.

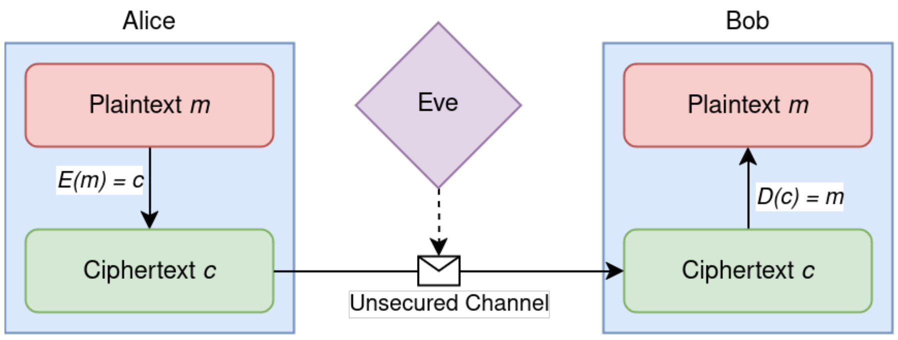


Figure 3.4: Two-party communication using a cryptographic scheme (based on [100]).

HASH FUNCTION
f63e68ac0...

Figure 3.5: Exemplary workings of a hash function [100].

3.3.2 Hash Functions

According to Wong [100], "hash functions are everywhere in cryptography". A hash function takes any data as input, such as files, videos, or messages, and produces a string in return, called a *digest*. Figure 3.5 shows an example of a hash function. A hash function is deterministic, it produces the same output whenever given the same input. While the input can be of any size, even empty, the output is of fixed length. Hash functions must provide three security properties in order to be of cryptographic use:

1. **Pre-image resistance:** This property, also referred to as *one-wayness*, ensures that it is computationally infeasible to recover the input if given an output.

2. **Second pre-image resistance:** If given a hash input and the resulting digest, it should be computationally infeasible to find a different input that produces the same digest.

3. **Collision resistance:** This property states that it is computationally infeasible to produce two different inputs that generate the same digest.

Bishop [18] provides this practical example: assume Alice wants to send Bob a message such that Bob can verify that the message has not been altered. Alice applies a hash function on the message to generate a digest. Alice then sends the message and the digest to Bob using two different channels. Bob can generate a digest from the message and compare it to the digest he received. If both digests match, Bob can assume the message has not been altered.

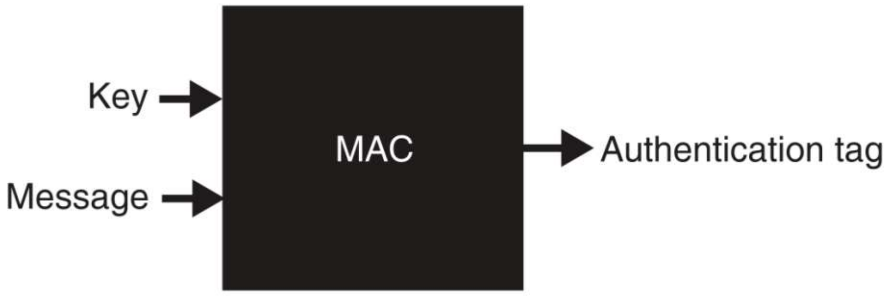


Figure 3.6: Exemplary workings of a MAC function [100].

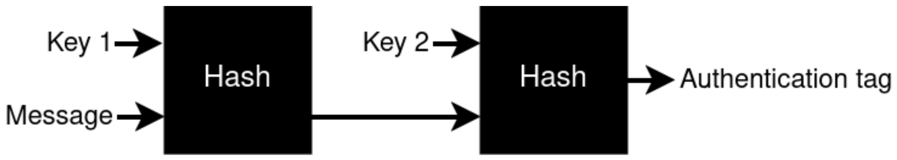


Figure 3.7: Exemplary workings of a HMAC function (based on [100]).

3.3.3 Message Authentication Codes

Wong [100] argues that there can be no confidentiality and authentication without using secret keys. By mixing a hash function with a secret key, the resulting function is called a Message Authentication Code, sometimes also *keyed hash* [31]. Wong goes on to state that a Message Authentication Code (MAC) can be used to protect the integrity of data. Similar to a hash function, a MAC takes as input any data and additionally also a secret key and produces as output an *authentication tag*. A MAC can be thought of as a personalised hash function where only the participants in possession of the secret key can produce the authentication tag. See Figure 3.6 of an example MAC function.

A standardised MAC that is widely used in practice is the hash-based Message Authentication Code (HMAC) [100, 83], which was invented by Bellare, Canetti, and Krawczyk [13]. The main idea behind the HMAC is to use hash functions to create a MAC algorithm, as stated by Banoth and Regar [10]. The general HMAC algorithm is as follows [100]:

1. Create two new secret keys from the secret key.
2. Concatenate the first secret key with the message and hash the result.
3. Concatenate the second secret key with the result from the previous step.

By chaining two hash calculations the risk of collisions is reduced significantly [85]. See Figure 3.7 for an example of an HMAC algorithm.

Blake2, which is used by WireGuard, is a hash function designed by Aumasson et al. [8]. The *Blake2* hash function can be adopted for different use cases:

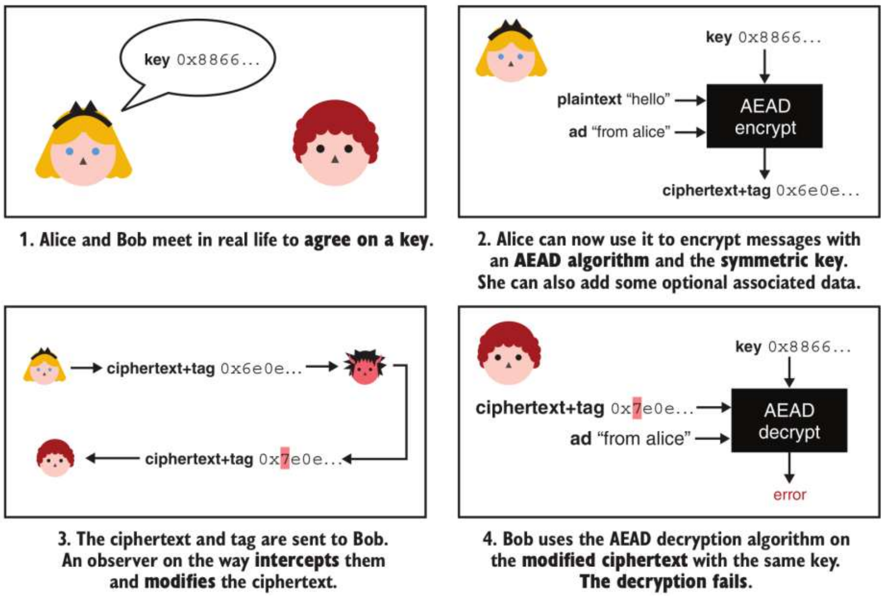

key 0x8866...
1. Alice and Bob meet in real life to agree on a key.
key 0x8866...
plaintext "hello"
ad "from alice"
AEAD encrypt
ciphertext+tag 0x6e0e...
2. Alice can now use it to encrypt messages with an AEAD algorithm and the symmetric key. She can also add some optional associated data.
ciphertext+tag 0x6e0e...
ciphertext+tag 0x7e0e...
3. The ciphertext and tag are sent to Bob. An observer on the way intercepts them and modifies the ciphertext.
key 0x8866...
ciphertext+tag 0x7e0e...
ad "from alice"
AEAD decrypt
error
4. Bob uses the AEAD decryption algorithm on the modified ciphertext with the same key. The decryption fails.

Figure 3.8: Exemplary workings of an AEAD construction [100].

1. *Blake2s* is a hash function specifically designed for small architectures.
2. *Keyed-Blake2s* is a keyed version of the hash function used for MACs.
3. *HMAC-Blake2s* is an HMAC construction using *Blake2s*.

3.3.4 Authenticated Encryption with Associated Data

Wong [100] argues that history has shown that developers are having a hard time applying cryptography in the real world. Efforts have been made to standardise all-in-one constructions that should simplify the use of encryption. Authenticated Encryption with Additional Data (AEAD) is such an all-in-one construction.

Authenticated Encryption (AE) in its broader sense refers to the combination of encryption with integrity protection and in its narrower sense only to those modes that use MAC to protect the integrity [83]. Because the MAC calculations can be done independently of the encryption it is possible to include additional *associated data* besides the ciphertext, resulting in AEAD [83] which is the most current way of encrypting data [100]. This associated data is optional and can contain metadata or be empty and will not be encrypted [100]. See Figure 3.8 for an example AEAD.

A widely adopted AEAD construction is *ChaCha20-Poly1305* [100]. It combines the *ChaCha20* stream cipher with the *Poly1305* MAC, both designed by Bernstein [16]. A *stream cipher* produces a series of random bytes with the same length as the plaintext and generates a ciphertext by XORing the plaintext with the random bytes [100].

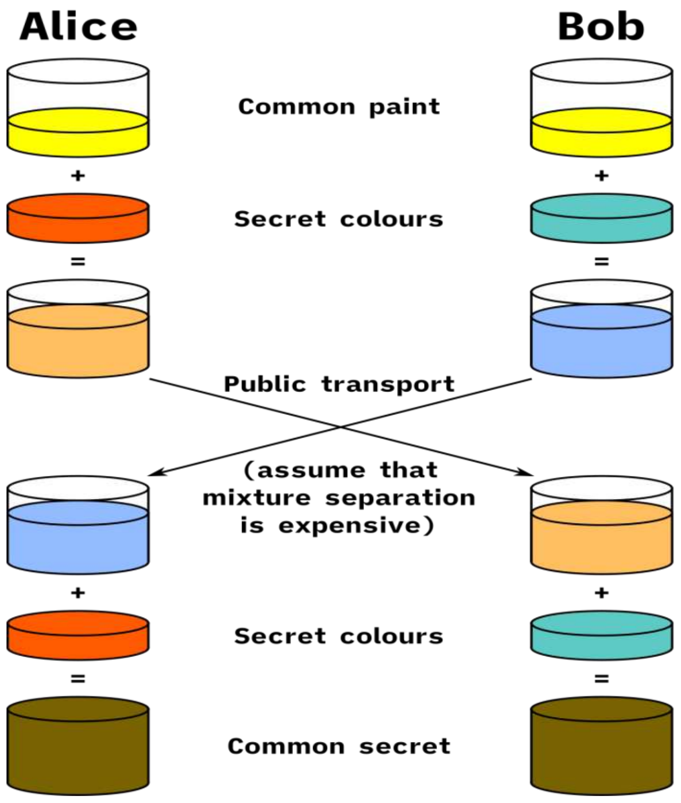


Figure 3.9: Illustration of the Diffie-Hellman key exchange [10].

3.3.5 Diffie-Hellman Key Exchange

In a *symmetric* encryption scheme, messages between sender and receiver are secured by a single secret key which is assumed that only those two parties know [83]. If Alice and Bob want to communicate using symmetric encryption, one of them needs to create the secret key and send it to the other. Sending a secret key over a network where an attacker is *passively* snooping in on the transmission would result in the attacker acquiring the secret key and being able to decrypt all messages [100]. A *key exchange* is an *asymmetric* cryptographic primitive that allows two peers to agree on a shared secret [100]. The key exchange starts with the participants Alice and Bob each generating a key pair, consisting of a private (or secret) key and a public key [100]. Alice and Bob can then exchange their public keys and use them together with their respective private keys to derive the same shared secret [100]. The first practical asymmetric key exchange was proposed by Diffie and Hellman [27] and is commonly referred to as Diffie-Hellman Key Exchange (DHKE) [100, 83] or Diffie-Hellman Key Agreement [85, 67]. The concept behind DHKE is illustrated in Figure 3.9, where the common colour represents the public keys of Alice and Bob and the secret colours represent the private keys.

The DHKE algorithm can be implemented using different kinds of mathematical groups. *Elliptic curves* (EC) are a type of curves studied in mathematics and can be used to implement DHKE. A DHKE using elliptic curves is referred to as Elliptic Curve Diffie-Hellman Key Exchange (ECDH) [100].

3.3.6 Key Derivation Function

A Key Derivation Function (KDF) is a common pattern in cryptography that can derive several secrets from one initial secret [100]. Krawczyk [55] states that the goal of a KDF is to derive a cryptographically strong key from an initial random keying material, such as from the result of a DHKE. The notion of a *cryptographically strong* key means that the key cannot be distinguished from a random string of the same length in a feasible time. The most popular KDF is the HMAC-based Key Derivation Function (HKDF) [83] introduced by Krawczyk [55] and is built on top of HMAC [100].

3.4 Software Testing

According to Mishra and Mohanty [69], software testing fulfils a variety of objectives. The main objective of software testing is the *verification* and *validation* of software. *Verification* is concerned with checking whether the software meets its specifications. *Validation* checks whether the software meets its user needs.

The performance of a software system, such as a VPN, is determined both by a specification and user needs. O'Regan [72] points out that performance testing is one of multiple test types performed during the development of a software system.

This section will present the fundamentals of software testing. Section 6 will then go into more detail about performance testing.

3.4.1 Motivation and Limits

As Bernhart and Breiteneder [15] point out, no software is without defects. This is why, according to Leloudas [56], software testing is essential to ensure the quality of a software system and that it performs as expected.

Subramanian et al. [89] state that software testing as a separate entity evolved later in the history of software development. In the beginning, testing was limited to debugging to develop a bug-free product. However, the absence of bugs does not mean that the software will meet the expected requirements. Over time, testing evolved into a way of thinking, identifying testing areas of interest and what can and cannot be tested.

Some limiting factors restrict the results of software tests. According to Leloudas [56], a lack of clear requirements and time constraints are common challenges in software testing as well as a shortage of skilled testers.

O'Regan [72] states that the test manager provides crucial recommendations in the decision of whether a software system should be released or not by highlighting any risks that are associated with a software product. Software development is deadline-driven, missed deadlines can lead to a compressed testing schedule. This in turn may lead to a shortened testing cycle, which could result in insufficient data to make an informed decision about the state of the software system.

Software testing is tasked with determining if the functionality of a software product is correct and complete. However, software testing cannot prove such claims [15].

3.4.2 Functional- and Non-Functional Testing

There are two common testing types defined in the literature, *functional* and *non-functional* testing (see, for example, [56, 15]).

Functional testing is concerned with whether a software product meets the user's requirements and works as expected [56]. The primary focus lies on the product's functionality [56] and "what" the system should do [88]. The main objective is to check the functional verification of the product [15], such as the *correctness*, *appropriateness*, and *completeness* [88]. This type of testing is most commonly associated with testing [88].

Non-functional testing covers everything not regarded as plain functionality. This includes but is not limited to, the performance, security, and usability of a product [56]. It checks "how" a system behaves [88]. Non-functional testing should be done at the earliest stage and all test levels [88]. The model of quality by ISO Central Secretary [47] forgoes the usage of the non-functional category and defines nine quality characteristics for a product: functional suitability, performance efficiency, compatibility, interaction capability, reliability, security, maintainability, flexibility, and safety.

Testing of the performance efficiency will be further discussed in Section 6.

3.4.3 Test Planning, Analysis and Design, and Execution

O'Regan [72] states that the quality of software testing relies on the maturity of the testing process. A good test process must include test planning, analysis and designing, and execution. Stapp, Roman, and Pilaeten [88] argue that the test process may or may not be formally defined.

Test planning: According to Leloudas [56], test planning is an essential activity in software testing. It involves creating a comprehensive test plan, starting with the definition of *testing objectives*. A *testing objective* defines the goals and outcomes that testing should achieve. Testing objectives can vary, but one of the main objectives is ensuring the quality of the software product. Defining the scope of a test is critical, as it identifies the areas and functions of interest and critically also the boundaries. Test planning involves selecting an appropriate testing approach and technique. Additionally, the identification and preparation of test data is also part of the test planning phase. Stapp, Roman, and Pilaeten [88] state that during the planning phase it is determined "what to test".

Test analysis and design: Stapp, Roman, and Pilaeten [88] state that during the analysis and design phase the question of "how to test the system" is being addressed. It starts by transforming the test objectives of the planning phase into *test conditions*. *Test conditions* are any kind of feature that can be checked during testing and typically do not contain expected results. *Test cases* and *test data* can then be derived from the *text conditions*. The high-level test cases are then implemented into concrete *test procedures* which are grouped into *test suites*. The *test environment* needs to be built during this phase.

Test execution: According to Stapp, Roman, and Pilaeten [88], during this phase the test suites are carried out. The version data of test items, test tools, and other testware must be registered. Tests are carried out manually or automated. The obtained test results are compared to the expected results. The test results are analysed to determine their cause and found defects are reported. The execution of the test is logged.

The basic test process outlined above was followed during the performance tests discussed in Section 7.

3.5 Statistics

Test data sets collected during software testing might make it necessary to use statistical analysis methods to present and comprehend the underlying information. As stated by Molyneaux [70] "statistical analysis lies at the heart of all automated performance test tools". This section will introduce the fundamental notions for analysing and displaying data sets.

According to Levine and Stephan [57], a *variable* is a characteristic of an item. Variables can be either *categorical* or *numerical*. While values of categorical variables are selected from a fixed list of categories, numerical variables are counted or measured. A counted numerical variable is called *discrete*, a measured numerical variable *continuous*.

3.5.1 Statistical Measures

According to Levine and Stephan [57], most numerical variables tend to cluster around a specific value that describes the central tendency of the variable. There are two common methods of identifying this property of a variable, the *mean* and the *median*. Fahrmeir et al. [33] explain that the mean and median characterise the centre of a data set distribution.

The *mean* is defined as the sum of all data values divided by the number of data values. Because the mean uses all data values in a set it can be distorted by individual extreme values [57].

The *median* is defined as the middle value of an ordered set of data values when the number of data values is odd. In case the number of data values is even, the mean of the

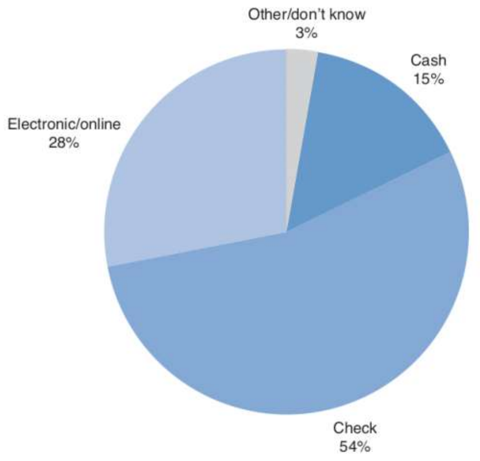


Figure 3.10: Example of a pie chart [57].

middle two data values is considered the median. The median is not affected by extreme values [57]. At least 50% of values of the data set are smaller or equal to the median, and at least 50% of values are equal or larger than the median [33].

While mean and median describe the central tendency of a data set, *quartiles* describe the relative position of a data value to the other values of a numeric variable, as discussed by Levine and Stephan [57]. Quartiles split a data set into four equal parts such that 25% of the ordered data values are smaller than the first quartile and 75% of the data values are smaller than the third quartile. The second quartile is the same as the median.

3.5.2 Displaying Statistical Results

Wilke [99] argues that the primary aim of data visualisation is to convey data accurately and must not be misleading. Yet, good visualisation can enhance the message of the visualisation. Thus, data visualisation should aim to be aesthetically pleasing. Wilke [99] acknowledges that aesthetic aspects are subjective and can vary over time. The plots presented in this section will be used to create aesthetically pleasing and accurate figures for performance test results.

Pie charts can be used for categorical variables. A pie chart presents the amount of each category as wedges of a circle where the angle of the wedge, and thus its area, is proportional to the amount [33]. Wilke [99] argues that pie charts work best to visualise simple fractions in small datasets. Figure 3.10 presents an example pie chart of a survey on how adults pay their monthly bills. The answers were grouped into the categories "Cash", "Check", "Electronic/online", and "Other/don't know".

A *bar plot* is another way to present categorical variables. For each category, a separate bar is drawn typically on the horizontal axis with the height of the bar representing the amount of that category in the data set [33]. In a *grouped bar plot* a group of bars is drawn on the horizontal axis based on one category and the height for each bar is

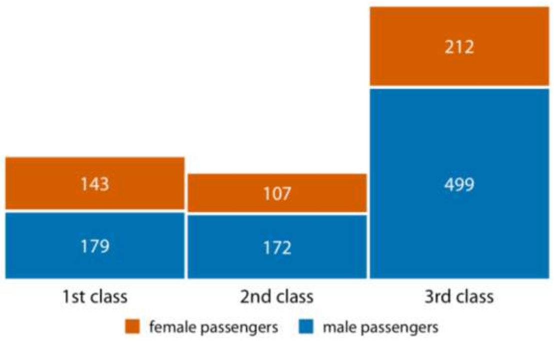


Figure 3.11: Example of a stacked bar plot [99].

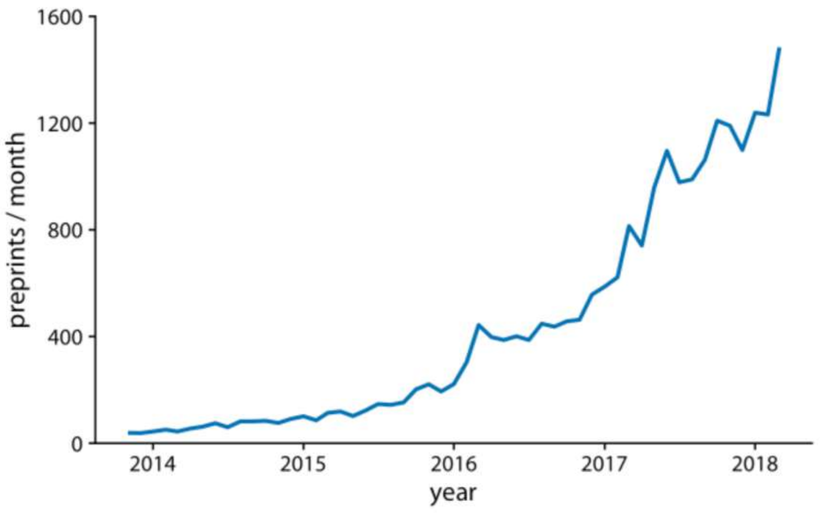


Figure 3.12: Example of a time-series plot [99].

determined by another category [99]. By stacking the bars instead of drawing them side by side, the resulting plot is a *stacked bar plot*. Wilke [99] argues that a stacked bar plot is useful when the sum of the amounts of each bar are meaningful and that this plot “works well for the visualisation of many sets of proportions or time series of proportions”. Figure 3.11 presents an example of a stacked bar plot on the number of passengers on the Titanic. The gender of the passengers was used as a category.

A *time-series plot* can be used to present numeric variables. The time-series plot conventionally presents units of time on the X-axis and units of the variable on the Y-axis [57]. This plot is used to visualise temporally ordered data [99]. Figure 3.12 presents an example of a time-series plot. The data presents the monthly submissions to the preprint server bioRxiv.

The *boxplot* is a way to visualise the shape of a set of data values. According to Levine and Stephan [57], a boxplot is a five-number summary of a data set, consisting of the

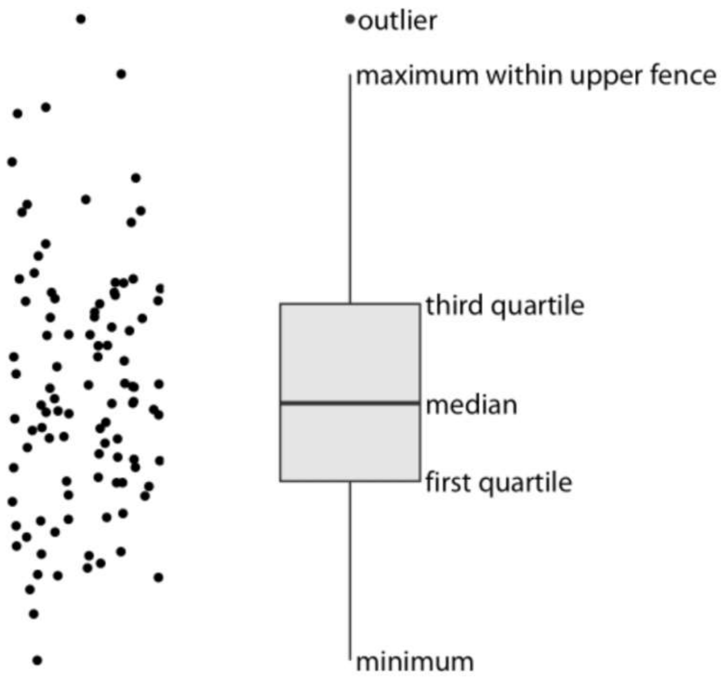


Figure 3.13: Example of a boxplot [99].

smallest value, the first quartile, the median, the third quartile, and the largest value. These five values are typically plotted as horizontal lines, connected by a vertical line, with the first quartile, median, and third quartile forming a box. Wilke [99] states that the first and last value of the boxplot could, instead of the smallest and largest value, extend to the maximum or minimum values that fall within 1.5 times the range of the box. Data values lying outside this range are referred to as *outliers*. Figure 3.13 presents an example box plot. On the left is a random cloud of points and on the right is the corresponding boxplot.

Wilke [99] argues that a more modern presentation of a data set shape is the *violin plot*, which provides a more nuanced picture. The violin plot begins at the minimum data value and stops at the maximum data value. The thickness of the plot corresponds to the density of the data set. Figure 3.14 presents an example violin plot. On the left is a random cloud of points and on the right is the corresponding violin plot.

As pointed out by Matejka and Fitzmaurice [64], different distributions can result in the same boxplot. For this reason, this thesis will make use of both, boxplot and violin plot. While the boxplot provides a quick overview of the underlying distribution, the violin plot can provide additional insight into a distribution.

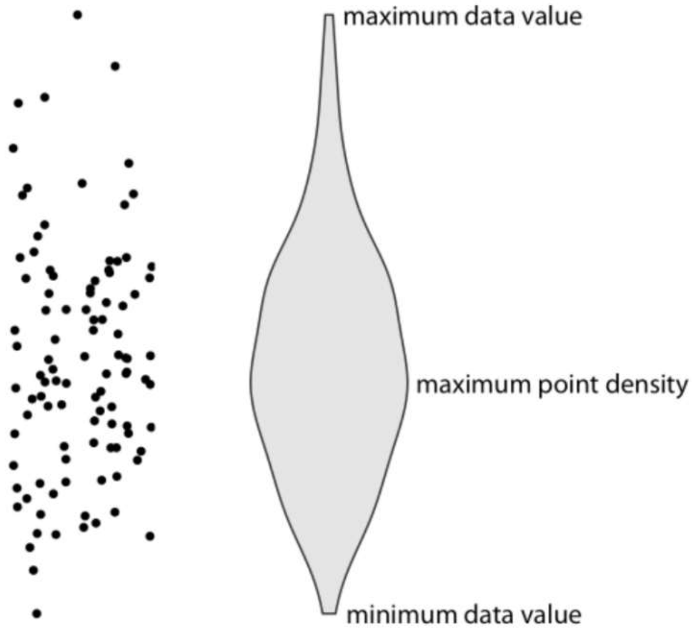


Figure 3.14: Example of a violin plot [99].

CHAPTER 4

Hardware Security Modules

This thesis will measure the performance of a VPN that is using a HSM in order to determine if this setup is capable of serving in a high-performance environment. This chapter will discuss what an HSM is in Section 4.1, Section 4.2 will discuss how keys are managed and Section 4.3 issues about physical security. Section 4.4 will present methods of scaling HSMs that can be found in the literature. Finally, Section 4.5 will present two HSMs as examples used during performance testing.

4.1 Purpose and Use Cases

A broad definition of an HSM by Mavrovouniotis and Ganley [65] is that an HSM is "any hardware device, with some level of tamper-resistance, which is used for cryptographic processing". A more detailed definition of an HSM is provided by Sommerhalder [86], stating that HSMs are specialised devices that perform cryptographic operations. They can use a random number source to generate public-private key pairs, manage keys, and store them. Additionally, they can be used to do encryption, decryption, and hashing. Some devices provide tamper-proofing features such as logging, and alerting mechanisms.

A comprehensive description of HSMs by Kamaraju, Ali, and Deepak [51] states that an HSM is a hardened physical computing device that specialises in key creation, key storage and execution of cryptographic processes. HSMs can be used in the form of a plug-in card that is used directly inside a device or as a network-accessible device. The aim is to provide a trusted computing platform that protects confidential materials. Kamaraju, Ali, and Deepak [51] go on to argue that the separation of cryptographic operations from business and database logic increases security.

HSMs are typically deployed in highly security-sensitive environments. In order to ensure that HSMs demonstrate a certain level of quality these devices commonly comply with one or more standards [65]. These standards specify security requirements that cover the

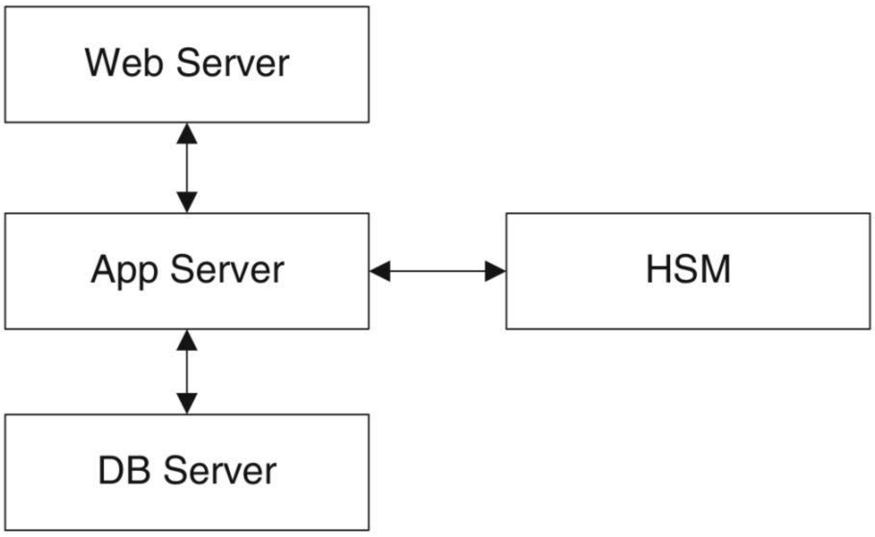


Figure 4.1: Example 3-tier architecture using an HSM [65].

design and implementation of an HSM. Examples of such standards are FIPS 140 [87], Common Criteria [25], or PCI-HSM [24].

As discussed in Section 1.1, HSMs can be used in a variety of situations and use cases. They are used for protecting personal data such as health records, for bulk encryption in satellite broadcasting, and as trusted third-party services [65]. Their primary use is for key management and payment [86]. Figure 4.1 presents a typical 3-tier architecture containing an HSM. Data received by the web server is passed to the application server, which encrypts the data with the help of the HSM and passes it on to the database server for storage.

4.2 Key Management

Kamaraju, Ali, and Deepak [51] state that security services such as encryption and decryption of data rely on cryptographic keys. Managing these keys securely is of importance. Tasks include key generation and distribution, storage and secure usage, and lifecycle management such as key rotation, backup, revocation, suspension, and deletion.

Mavrovouniotis and Ganley [65] argue that cryptography relies on the protection and proper use of keys. All keys stored inside the HSM should be backed up. Secret keys stored by the HSM must never appear in plain outside the confines of the HSM. Aside from protecting the confidentiality of secret keys, the integrity must also be protected. To this end, attackers must not be able to modify or misuse a key.

Mavrovouniotis and Ganley [65] state that there are two methods for protecting keys used by an HSM:

1. **Storing all keys inside the HSM:** All keys are stored inside the HSM. When using the HSM a pointer to the key to be used must be provided in the command. This method has two drawbacks. First, in case the HSM loses its keys all keys must be reloaded into the HSM. Second, key management becomes more difficult if multiple HSMs are used.

2. **Storing a single master key inside the HSM:** A single master key is stored inside the HSM. All other keys are encrypted with the master key. The encrypted keys are stored in a key database accessible to applications.

4.3 Physical Security

According to Mavrovouniotis and Ganley [65], physical security of an HSM device is a crucial aspect of providing high-grade security. An HSM contains a tamper-resistant core that contains all sensitive components. All cryptographic processing is carried out in the core system. The core system is provided with battery-backed volatile memory for the storage of plaintext cryptographic keys. Tamper-resistance can be achieved by wrapping the core system around with a fine electronic mesh and encasing it all in epoxy resin. An attacker trying to penetrate the resin would likely break the mesh. In case the core system detects such a physical attack, the secure memory will be deleted.

Physical security around the device needs to be strict. An HSM is typically locked inside a secure cabinet, located within a high-security area. The doors of the secure cabinet can be secured with dual physical control, meaning they need two controls to be opened. Controls could be a key, number combination, or a biometric key.

4.4 Scaling Methods

Two general ways to scaling resource can be distinguished, according to Sehgal, Bhatt, and Acken [84]:

- **Vertical Scale (scale up):** With this scaling method, the resources of existing systems are increased, for example by adding memory, Central Processing Unit (CPU) cores or hard disk space. Another option would be to replace existing systems with systems having more resources.

- **Horizontal Scale (scale out):** This scaling method adds systems of similar or the same size to the existing system to increase overall available resources. The applications must be able to run in a distributed manner in order to be able to scale in this way.

On-premise HSMs have only limited scalability, according to Han et al. [43]. Because all operations that require the secret keys managed by the HSM have to be carried out inside the HSM, the HSM becomes a performance bottleneck of the overall system.

Han et al. [43] propose a solution where a single HSM works in tandem with multiple software Key Management Systems (KMSs). In their solution the HSM is storing a root secret key that derives, rotates, and revocates derived keys for the KMSs. The KMSs handle user requests such as encryption and decryption by using the derived keys.

4.5 Examples of Hardware Security Modules

This section will present two examples of real-world HSMs from two different manufacturers. These HSMs were used during the performance tests discussed in Section 7 and will be presented in more detail in the following sections. These devices will be referred to as HSM-1 and HSM-2 for pseudonymity. While HSM-1 is more of an entry level device, HSM-2 represents a high-end device.

4.5.1 HSM-1

The HSM-1 is a network-attached peripheral device. It provides cryptographic services as a shared network resource. This device is FIPS 140-2 and Common Criteria EAL4+ certified. The following abstract of the HSM-1 data sheet provides some details about the performance, given in transactions per second (tps), and the supported cryptographic primitives.

- **Rivest-Shamir-Adleman (RSA) performance:** The data sheet states a performance of 430 tps for RSA-2048 and 100 tps for RSA-4096.
- **Elliptic curve cryptography performance:** The data sheet provides a performance value of 680 tps for the P256 curve.
- **Supported algorithms:** This HSM supports a wide range of symmetric and asymmetric primitives as well as hash functions. Of importance for this thesis is the support of ECDH, which this device supports.

4.5.2 HSM-2

The HSM-2 is a network-attached peripheral device. This device offers the maximum performance in this producer's product line. The device is certified according to FIPS 140-2 Level 3, Common Criteria EAL4+, and NATO Approved for Use up to Restricted. The following abstract of the HSM-2 data sheet provides some details about the performance and the supported cryptographic primitives.

- **RSA performance:** The data sheet states a performance of 10000 tps for RSA-2048 and no values for RSA-4096.
- **Elliptic curve cryptography performance:** The data sheet provides a performance value of 22000 tps for the P256 curve.

- **Supported algorithms:** This HSM supports a wide range of symmetric and asymmetric primitives as well as hash functions, similar to HSM-1. This device supports ECDH.

CHAPTER 5

Virtual Private Networks

The WireGuard protocol used in this thesis is a VPN protocol. VPNs are used to connect similar networks over a dissimilar middle network and enterprises use VPNs to supplement their WANs [80]. Section 5.1 will discuss the motivation of using a VPN, provide useful definitions, and discuss the limitations of VPNs. Section 5.2 will discuss the general underlying concept of VPNs. Section 5.3 will present some common use cases of VPNs. Section 5.4 will discuss the WireGuard protocol in detail.

5.1 Motivation, Definitions, and Limitations

Companies might have offices spread over many cities and countries. In order to connect these offices, companies used to lease dedicated lines between their office locations [91]. These leased lines provided a secure and private connection [80] but they were expensive [91]. With the rise of the Internet, connecting offices over the Internet became a more interesting and inexpensive option [80]. Additionally, the Internet allows mobile users, such as field staff, to connect to the corporate network [80]. However, normal TCP/IP traffic over the Internet is plaintext allowing every party with access to the packet stream to read, alter, or drop packets [97]. This would violate security goals such as confidentiality, integrity, and authentication (see Section 3.1.1). A naive approach would be to simply encrypt packets. However, such an approach would break existing protocols as they rely on plaintext header fields to allow for packet processing [97] (see Section 3.2.2).

Tanenbaum and Wetherall [91] define a *private network* as a network that is built from company computers and leased telephone lines. van Oorschot [97] takes the definition of a *private network* a step further by defining it as a "network intended for access only by trusted users, with security (e.g., confidentiality, integrity) relying on a network architecture providing physical isolation". This definition stresses that the physical isolation of the leased line provides the desired security goals. van Oorschot [97] goes on to define a *VPN* as a private network that is not secured by physical isolation but by the use of encrypted *tunnels* and special-purpose protocol software and hardware. Sadiku and Akujuobi [80] state that a VPN simulates a private network over a public network. It allows multiple devices to communicate with each other, using a combination of hardware and software. It is private because it is inaccessible to unauthorised parties and because the routing and addressing plans are independent of the public network.

At this point, it should be noted that the assumption that only trusted users have access to a private network may no longer be adequate. Finney and Kindervag [34] argue that the common trust model separates a network into an untrusted side, typically the Internet, and a trusted side. While the untrusted side is the focus of security considerations, the trusted side does not receive any real security considerations. Finney and Kindervag [34] state that almost all data breaches are an exploitation of this broken trust model. Garbis and Chapman [35] state that *zero trust* security follows the principle of least privilege for networks and applications where trusted users should only have the minimum necessary access. VPNs can provide an additional layer of security in a zero trust model.

Carmouche [21] argues that VPNs exist to protect data that is transmitted between two networks. To this end, a VPN must meet the four goals of confidentiality, integrity, non-repudiation, and authentication. These security goals are discussed in Section 3.1.1.

Some challenges arise when using a VPN:

- The encryption and decryption of data, as being done by a VPN, is a computing-intensive and expensive task [80].
- A VPN requires a certain quality of service, e.g., a minimum guaranteed bandwidth for connections between end points [80].
- Incompatibility issues can arise if devices from different vendors are being used [80].
- Because of the ubiquitous use of VPNs in the corporate world, it has become a point of interest for attackers. Gaining access to a corporate VPN can open a backdoor to the network [80].
- Network-based monitoring becomes more difficult when packets are being encrypted [97].

5.2 Tunnelling

According to Tian and Gao [92], the basis for VPNs is encryption and *tunnelling*.

A typical case of tunnelling, given by Tanenbaum and Wetherall [91], is the case where two networks, such as two remote offices, are using the same network protocol but are connected with a network that uses another protocol. By *encapsulating* the network protocol message of the source network inside the network protocol used by the intermediate network, the message can travel across the intermediate network. Upon arrival at the destination network, the intermediate network protocol is removed from the message and the message can travel on to its destination. The path through the intermediate network can be seen as a tunnel where the messages enter one side of the intermediate network, travel across the intermediate network, and can only exit at the destination network.

In contrast to the standard network protocol design where protocols lower in the OSI layer carry payloads of protocols of higher layers (see Section 3.2.2) tunnelling may involve one protocol carrying a protocol of the same layer [97]. Tunnels do not necessarily provide security features [92]. A VPN is an encrypted tunnel [80].

According to Tian and Gao [92], the creation of a VPN requires three types of protocols:

1. A passenger protocol: This protocol contains the original data to be transported between two VPN connected networks. The IP protocol discussed in Section 3.2.4 is an example of such a protocol.

2. An encapsulation protocol: This protocol encapsulates the passenger protocol. It defines how the passenger protocol is being encapsulated and whether it is encrypted.

3. A carrier protocol: This network transport protocol must be supported by the transit network. It carries the encapsulation protocol over the intermediate transit network. One such protocol would be the Point-to-Point Protocol (PPP).

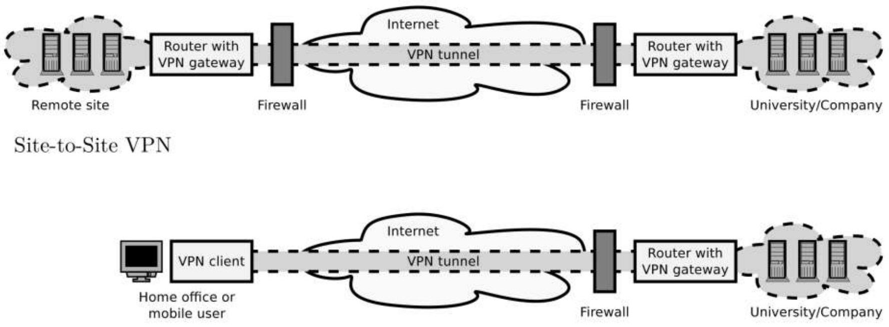
Although tunnelling can be implemented at virtually any layer it is commonly implemented at Layer 2 and Layer 3 of the OSI model (see Section 3.2.2) [80].

Layer 2 VPNs: According to Baun [12], a layer 2 VPN can be designed as both a site-to-site and a remote access VPN. In layer 2 VPN the VPN clients and VPN gateways are encapsulating the data packets. An example of a layer 2 VPN protocol is the L2TP. L2TP is based on two older layer 2 protocols [44]. It is considered to be lightweight and robust [92] and can be used on non-IP-based networks [80].

Layer 3 VPNs: A layer 3 VPN operates at the network layer of the OSI model. Such a VPN can occur over any TCP/IP network, such as the Internet [44]. The IPsec is a typical protocol for layer 3 VPN [12]. IPsec is a popular layer 3 tunnelling protocol that operates directly on top of the IP protocol [92]. It extends IP to provide support for security features just as authentication and integrity [44].

5.3 VPN Applications

VPNs can be used in different application scenarios.



Site-to-Site VPN

Remote Access VPN or End-to-Site VPN

Figure 5.1: A Site-to-Site VPN (top) compared to a Remote Access VPN (bottom) [12].

Remote Access VPN: The workforce is becoming more and more mobile making it necessary to provide employees with remote access to the corporate network [80]. In order to allow field staff and employees working from home to connect to the corporate network, a *remote access* VPN can be set up [44] (also referred to as *end-to-site* VPN [12]). The VPN client establishes connection to a VPN gateway [12] or a router [44].

Site-to-Site VPN: A *Site-to-Site* VPN connects two locations, such as a branch office and headquarters, via a public network [44]. It can also be used to connect the networks of two different companies [44]. The hardware on which the VPN terminates can be a router with VPN capabilities, a firewall, or a VPN gateway [44] (see Section 3.2.3 on routers and gateways). A Site-to-Site VPN is used to either supplement or replace leased lines [80]. Figure 5.1 presents the difference between a remote access VPN and site-to-site VPN.

Client-to-Site VPN: Tian and Gao [92] discuss that a Client-to-Site VPN is similar to a Remote Access VPN by providing individual users with secure remote access to a network resource over the Internet. Thus, the terms Remote Access and Client-to-Site are often used interchangeably. The main difference is that a Remote Access VPN connects clients to the company-owned network and resources, while a Client-to-Site VPN typically provides access to a network resource hosted by a third party.

5.4 Case Example: WireGuard

This section will present key aspects of the WireGuard VPN protocol. WireGuard is a secure network tunnel devised by Donenfeld [29]. It is an encapsulation protocol that operates on layer 3 of the OSI, as stated by Abdulazeez et al. [1]. It is typically used in remote-access applications but can also be used in Site-to-Site scenarios.

WireGuard is an open-source VPN protocol focusing on high speed and simplicity, as stated by Donenfeld [29]. Due to its small code base, as noted by Dowling and

Paterson [30], WireGuard lends itself to making changes to the code that allow a connection to an HSM.

5.4.1 Design Principles

This section will present the main design principles that the WireGuard protocol is following.

Cryptokey Routing: According to Donenfeld [29], the fundamental principle of WireGuard is called *cryptokey routing*. It associates each peer with a list of allowed IP addresses. A *peer* is strictly identified by its public key. Public keys are points on the Curve25519. The *Curve25519* is an elliptic curve that is nowadays used by most applications in an ECDH [100] (see Section 3.3.5). The *cryptokey routing table*, therefore, contains a list of public keys, each associated with its own set of allowed IP addresses. The cryptokey routing table is used for incoming packets as well as outgoing packets. In the case of an outgoing packet, the table is consulted to find out which public key to use for encryption, by looking up which public key is associated with the destination IP of the packet. In the case of an incoming packet, after the packet is decrypted and authenticated, the packet will only be accepted if its source IP corresponds to the public key that was used during decryption. The benefit of this design is that all packets that are accepted are authentic. This routing algorithm used by WireGuard can be considered static (see Section 3.2.5).

Silence is a Virtue: Master and Garman [62] argue that WireGuard is a silent protocol. In case a WireGuard peer does not have messages to send out, it refrains from sending out keepalive messages by default. However, it is possible to change this default behaviour by enabling keepalive messages via a configuration option. WireGuard does not respond to unauthenticated packets. A packet is unauthenticated if it does not match an appropriate tunnel IP or associated public key.

Pre-shared Keys: WireGuard offers an option to mitigate future advances in quantum computing by allowing each pair of peers to add a 256-bit symmetric key to the encryption [30]. While WireGuard does not offer full post-quantum security, pre-shared keys provide an additional layer of encryption that can help mitigate the impact of quantum computing. Symmetric encryption is considered to be relatively secure to quantum attacks, as stated by Bernstein [17].

Denial of Service Mitigation: When receiving a handshake message, the responder needs to authenticate the message by computing a Curve25519 point multiplication. Even though Curve25519 is a fast curve, this multiplication is CPU intensive. This opens up a possibility of a resource exhaustion attack, targeting the availability of the system [29]. Lipp, Blanchet, and Bhargavan [60] state that the recipient of a handshake message can decide to not process a handshake message if it is under load and instead reply with a *cookie* message. The initiator can use the received cookie in the next message and have it be accepted. A cookie is the result of computing a MAC of the initiator's source IP using a secret key as the MAC key. This ties the sender of a message to an IP address.

Persistent Keepalive: WireGuard implements a keepalive mechanism that allows sessions to stay active. This mechanism is disabled by default. This keepalive mechanism allows peers to determine passively whether a connection has failed or was disconnected. Donenfeld [29] argues that every transport data sent, warrants some kind of response, whether that a response generated by the nature of the encapsulated packet or this keepalive message. If a peer has received a data message from another peer but does not have to send any packets itself, it will send out a *keepalive message*. A keepalive message is simply a zero-length encapsulated inner packet and can thus be easily distinguished from other messages. In case a peer has not received any transport messages for a certain time, it can be concluded that the secure session is broken.

5.4.2 Message Flows

This section will discuss how two WireGuard peers are communicating with each other. Foundations on cryptography relevant to this section can be found in Section 3.3. Figure 5.2 presents the general handshake message flow as well as how the handshake message flow could look in case the responder is under load. WireGuard uses four types of messages [29]:

- Handshake Initiation
- Handshake Response
- Cookie Reply
- Transport Data

According to Appelbaum, Martindale, and Wu [6], WireGuard is using IP as carrier protocol and UDP as passenger protocol for all messages (see Section 5.2).

Each peer maintains two kinds of keys that are used during the message exchange [29]:

- A static private key. This key is set once and used to authenticate one peer to another peer (see Section 5.4.1). The static private key is typically stored inside a configuration file. This thesis discusses the performance impacts if the static private key is stored inside an HSM.
- An ephemeral private key. This key is generated new for each session.

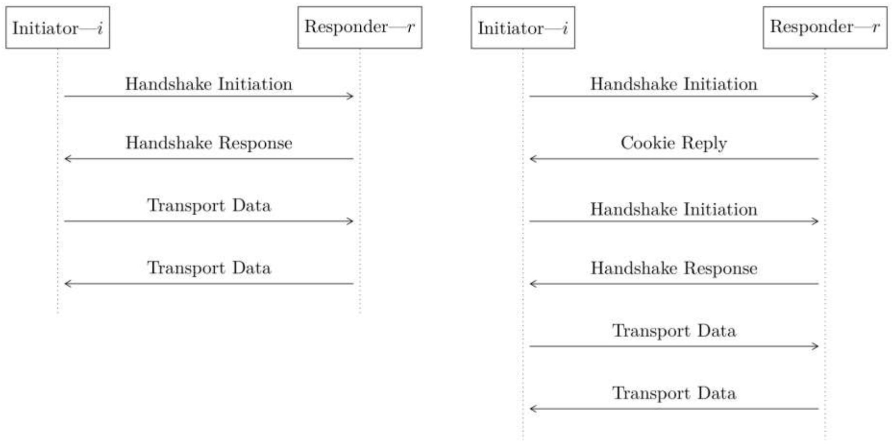


Figure 5.2: The WireGuard handshake message flow in general (left) and in case the responder is under load (right) [29].

Handshake Initiation: The first message sent by a peer is referred to as *handshake initiation*. This message consists of the following fields [29] as presented in Figure 5.3:

- **type:** This field contains the type of the message. In the case of the handshake initiation, this value is set to "1".
- **reserved:** This field is reserved to allow reading the type field as a 4-byte field.
- **sender:** This 4-byte field represents a peer and is generated randomly. Subsequent messages can be tied to the session created by this initiation message using this index.
- **ephemeral:** This field contains the ephemeral public key of the initiator. The Curve25519 private key is generated randomly and the corresponding public key is used for the ECDH (see Section 3.3.5).
- **static:** This field contains the result of a ChaCha20Poly1305 AEAD using the initiators static public key as input.
- **timestamp:** This field contains the result of a ChaCha20Poly1305 AEAD using the current time as input. This timestamp is included to prevent attackers from replaying handshake initiations.
- **mac1:** This field must always contain a valid MAC. Otherwise, the receiver will drop the message without responding to the sender. This follows the "silence as a virtue" property of WireGuard, as discussed in Section 5.4.1.
- **mac2:** This field is zero in case the initiator does not have a valid cookie available from the responder. Otherwise, this field contains a MAC using the valid cookie as MAC key.

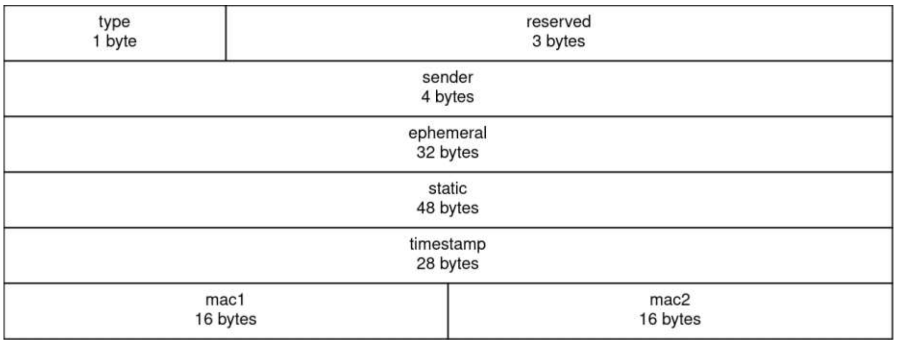


Figure 5.3: WireGuard handshake initiation message.

Handshake Response: The responder completes the ECDH by using the provided ephemeral public key from the initiator [30]. In case the static private key of the responder lies inside an HSM, this computation is done inside the HSM. After processing the handshake initiation message, the responder sends back the handshake response as presented in Figure 5.4. The message consists of the following fields [29]:

- **type:** This field contains the type of the message. In the case of the handshake response, this value is set to “2”.
- **reserved:** This field is reserved to allow reading the type field as a 4-byte field.
- **sender:** This 4-byte field represents a peer and is generated randomly. Subsequent messages can be tied to the session created by this initiation message using this index.
- **receiver:** This field is set to the peer index received in the handshake initiation.
- **ephemeral:** This field contains the ephemeral public key of the responder. The Curve25519 private key is generated randomly and the public key is derived from it.
- **empty:** This field contains the result of a ChaCha20Poly1305 AEAD using an empty string as input.
- **mac1:** This field must always contain a valid MAC. Otherwise, the receiver will drop the message without responding to the sender. This follows the “silence as a virtue” property of WireGuard, as discussed in Section 5.4.1.
- **mac2:** This field is zero in a handshake response.

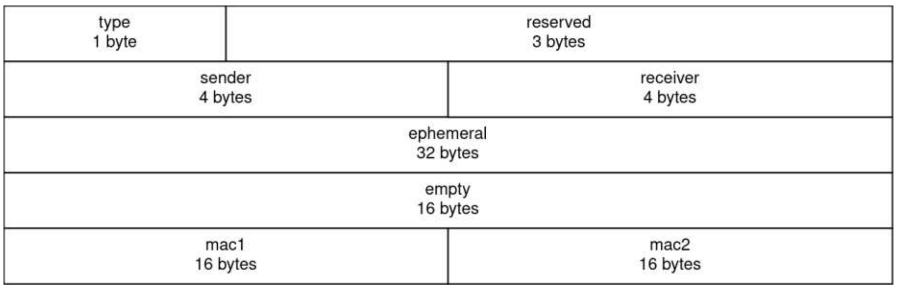


Figure 5.4: WireGuard handshake response message.

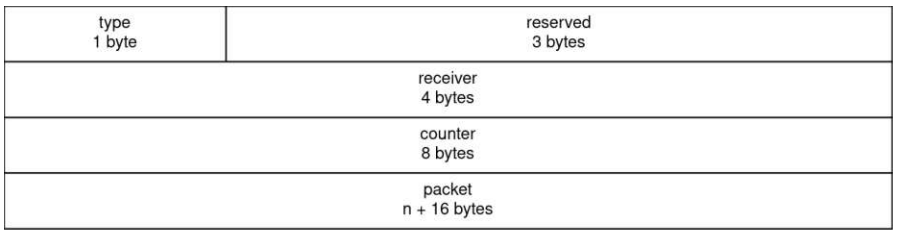


Figure 5.5: WireGuard transport data message.

Transport Data: After exchanging handshake messages, the sender and receiver can start exchanging transport data messages. Transport data messages contain encapsulated encrypted data. The message consists of the following fields [29] as presented in Figure 5.5:

- **type:** This field contains the type of the message. In the case of the transport data message, this value is set to "4".
- **reserved:** This field is reserved to allow reading the type field as a 4-byte field.
- **receiver:** This field is set to the peer index received in the handshake initiation or handshake response.
- **counter:** This message counter is used as nonce for the ChaCha20Poly1305 AEAD. The counter is incremented with each transport data message and serves to avoid replay attacks.
- **packet:** This field contains the encapsulated data encrypted using ChaCha20Poly1305 AEAD. This field contains n bytes from the encapsulated data plus a 16-byte authentication tag from the ChaCha20Poly1305 AEAD.

Cookie Reply: As discussed in Section 5.4.1, the receiver of a handshake initiation can decide to respond with a cookie reply instead of a handshake response in case the receiver is under load. The message consists of the following fields [29] as presented in Figure 5.6:

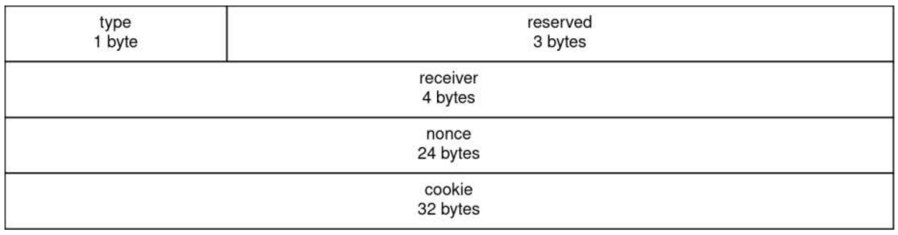


Figure 5.6: WireGuard cookie reply message.

- **type:** This field contains the type of the message. In the case of the cookie reply, this value is set to "3".
- **reserved:** This field is reserved to allow reading the type field as a 4-byte field.
- **receiver:** This field is set to the peer index received in the handshake initiation.
- **nonce:** This field contains random data. It is used in the creation of the cookie value.
- **cookie:** This field contains the cookie value. It uses the sender's source IP as well as a random secret value to create a cookie.

5.4.3 Rekeying and Rejecting

WireGuard has defined some constants and timeouts which are of interest for this thesis. This section will discuss the following constants and timeouts in more detail [29]:

- Rekey-After-Time and Rekey-After-Messages
- Reject-After-Time and Reject-After-Messages

Rekeying: WireGuard defines a constant and a timeout that relates to rekeying, the *Rekey-After-Time* timeout, and the *Rekey-After-Messages* constant. The *Rekey-After-Messages* constant defines the number of messages a peer can send before it has to create a new session by sending a new handshake initiation. This behaviour applies to both sides of a session, the initiator and the responder. The *Rekey-After-Time* timeout on the other hand applies only to the initiator of a session. It defines that the initiator will try to establish a new session after the timeout has passed by sending a new handshake initiation. Since the value for *Rekey-After-Time* is only 120 seconds and the value for *Rekey-After-Messages* is 2^{60} a rekeying likely occurs around every two minutes. The *RekeyTimeout* constant, set to 5 seconds, defines how long WireGuard will wait for a handshake response before sending a new handshake initiation.

Rejecting: In case the rekeying of a session is unsuccessful, WireGuard defines the *Reject-After-Time* timeout, and the *Reject-After-Messages* constant. The timeout *Reject-After-Time* is set to 180 seconds. In case the ephemeral key of a session is older than the timeout, WireGuard will not send or receive any more packets from this session. The constant *Reject-After-Messages* will do the same after $2^{64} - 2^{13} - 1$ transport data messages are exchanged.

CHAPTER 6

Performance Testing

Performance testing is concerned with evaluating the behaviour of a system under test. This thesis will carry out performance tests on a WireGuard VPN extended with an HSM. Section 6.1 will present definitions, discuss the goals and purpose of performance testing, and the challenges that come with performance testing. Section 6.2 will go on to discuss different performance testing types and strategies. Section 6.3 will discuss which metrics are commonly considered during performance testing. Section 6.4 will discuss how workload is generated in performance testing. Finally, Section 6.5 will present a general performance test setup.

6.1 Definitions, Goals, Purpose, and Challenges

No clear definition of *performance* can be found in the literature and the term *performance testing* is also defined differently in many sources. For example, Yorkston [101] states that *performance testing* is “testing to determine the performance of a software product”. Yorkston [101] goes on to provide a vague definition of *performance*, stating that “performance is a component of a user’s “good experience” and forms part of an acceptable quality level”. Yorkston [101] acknowledges that it is difficult to define “good” and “bad” performance and that it is the task of a performance engineer to define how performance is to be quantified and measured. Molyneaux [70] similarly argues that “performance really is in the eye of the beholder”.

Jiang and Hassan [49] provide a comprehensive review of existing definitions of the term *performance testing* and provide the following definition:

> [. . .] performance testing is used to measure and/or evaluate performance related aspects (e.g., response time, throughput and resource utilisation) of algorithms, designs/architectures, modules, configurations, or the overall systems.

Matam and Jain [63] take a similar approach by defining *performance testing* over the attributes that are being tested:

> Performance testing is the testing performed on a system or application to measure some of its attributes such as response time, throughput, scalability

The significance of performance testing is given by Bennett [14] who agrees with the above definition by stating that performance testing plays a critical role in ensuring that a software system delivers "optimal speed, responsiveness, scalability, and stability". Matam and Jain [63] add to this by arguing that a performant software system, such as a website, increases the profitability of a business by providing a better user experience, utilising fewer resources, and using resources more efficiently.

Matching to the definitions given above, Gregg [39] as well as Bennett [14] state that the goal of performance testing is to improve the end-user experience.

Long lists of challenges concerning performance testing can be found in the literature (see, for example, [39, 14, 70]).

- The performance of a software system is *subjective*. It is unclear what constitutes "bad" performance and at what point it is fixed [39].
- Due to the complexity of systems, performance can be a challenging discipline. This is especially true in cloud computing environments where it can be difficult to even find a starting point for analysis. Performance issues in complex systems can stem from the interactions between systems that perform well when analysed separately but perform poorly when put together, making the analysis even more difficult [39].
- Performance issues can have multiple contributing factors where multiple normal events result in a performance issue [39].
- Complex software systems often have many performance issues, sometimes even known to the developers. The important part is finding the performance issues that matter the most [39].
- Acquiring accurate data for performance testing can be challenging. Working with inaccurate data can lead to misleading results [14].
- The interpretation of performance results can be challenging. Knowledge of the goals and user expectations in the context of the project is important [14].
- Quantitative testing alone might not be sufficient to get the full picture. It might be sensible to supplement quantitative testing with qualitative tests such as user feedback [14].

- The iterative and dynamic nature of software development can pose a challenge to performance testing. Adapting performance testing strategies to the software development process can be beneficial [14].

- Creating realistic test inputs for performance testing can be challenging. Understanding real-world user usage patterns is crucial in creating adequate performance tests [14].

6.2 Testing Types and Strategies

Multiple different performance testing types can be found in the literature that can help with evaluating the performance of a software system. Yorkston [101] provides the following performance testing types:

- **Load testing:** Load testing evaluates the systems' performance under various workloads. The workload can range from anticipated realistic load [101] to peak load [56].

- **Stress testing:** Stress testing evaluates the system's performance under conditions at or beyond the anticipated limit. Stress tests are usually derived from Load tests. Stress testing aims to identify the breaking point of the system.

- **Scalability testing:** The goal of Scalability testing is to determine the system's ability to grow which can be tested in different ways. One approach is to increase the system's resources, either by scaling the system up or by adding more systems of the same specification and testing with the same workload. Another approach would be to simulate different levels of load and measure the system's response time and resource utilisation [56]. Doubling the load should result in double resource utilisation.

- **Spike testing:** Spike testing focuses on evaluating the system's ability to handle short and sudden bursts of peak loads. The system is expected to recover from a short spike in load and return to a ready state afterwards.

- **Endurance testing:** Endurance tests evaluate the system's performance over an extended period to determine if the system can maintain its performance [56]. The main difference to Load testing lies in the duration of the test. Endurance tests may execute for several hours up to days or even weeks.

- **Concurrency testing:** Concurrency testing forms a cornerstone of performance testing. Although a single user can generate load on a system, that load might not be enough to truly test a system. Concurrency testing aims to evaluate a system when a large number of users use it simultaneously.

- **Capacity testing:** In contrast to Stress testing, which is trying to cause a failure in the system, Capacity testing is trying to establish how many users the system can support at most. Capacity testing is typically carried out with a performance objective in mind, e.g., keeping RTT below a certain threshold. Results from these tests can help an organisation determine if an expected growth rate can be met.
- **Configuration testing:** Configuration testing is testing the same system using different configurations to determine how changes to the configuration affect its performance.
- **Comparative testing:** By comparing the performance of different systems or the same system in different configurations it is possible to determine which system performs the best.

Literature provides some insights into how to successfully set up performance tests:

- **Performance goals:** Defining performance goals is essential to align business and user expectations [14]. Results of performance tests can be compared to defined performance goals to assess the state of the software system [70].
- **Test environment:** It is important that the test environment closely resembles the production environment in order for the test results to be representative [14]. Ideally, the test environment would be an exact copy of the production environment, but this is rarely possible [70]. Performance testing should not be carried out in the production environment as this could negatively impact the performance and security of the system [63].
- **Baseline testing:** Establishing a baseline performance level of the system under normal conditions provides a reference point for future evaluations [14]. Baseline testing should be carried out without any activity on the system to provide a best-case measurement [70].

6.3 Performance Metrics

According to Bath et al. [11], it is necessary to understand which measurements and metrics are needed before carrying out performance tests. Performance goals can only be defined after performance metrics have been defined. Metrics will vary depending on the context of the software system.

General performance metrics discussed in the literature include throughput and delay which are also relevant network performance metrics, as discussed in detail in Section 3.2.6. Another performance metric typically discussed in literature is *resource utilisation* (see, for example, [101, 63, 14, 70]). The system may be restricted in the amount of resources it is allowed to use [70]. Utilisation is the ratio of used capacity to available capacity [63]. Resources of interest generally include CPU, memory, and disk usage [14].

A resource of interest for this thesis is the CPU. The CPU utilisation is measured as the time a CPU instance is busy performing work during an interval [39]. This utilisation is typically expressed as a percentage [39]. Gregg [39] argues that a high CPU utilisation is not necessarily a problem because performance does not degrade under high utilisation. Even further, a high CPU utilisation can be viewed as a good return on investment. Matam and Jain [63] point out that the system utilisation should not exceed 80% to allow the system to handle spikes in the load.

According to Bovet and Cesati [19], every Unix kernel keeps track of CPU activity. This activity can be monitored using different Unix tools, such as *top* or *uptime*. These tools present the CPU activity as *load average* relative to the last minute. A *load average* of 0 means there are no active processes, while a value of 1 means the CPU is 100% busy. The manual page of *uptime* [95] states that on a CPU system with 4 cores a value of 1 means the system was idle 75% of the time.

After performance test result data is gathered the results need to be analysed. Bath et al. [11] argue that analysis of raw test data can be misleading. The aggregation of the raw test data allows the data to be presented in a simpler and clearer way and the results of the tests to be communicated and reported in a simple form.

6.4 Workload Generation

In order to carry out performance tests, it is necessary to define the workload first. The definition and creation of workload for performance testing presents some challenges. Obaidat and Boudriga [73] argue that it might not be clear what level of detail is necessary for the workload and that identification of relevant aspects of the workload can be difficult. Weyuker and Vokolos [98] point out that the acquisition of test data can be difficult in itself. An operational profile created from the monitoring log files of a system similar to that of interest can be used as a basis.

Bath et al. [11] point out that performance test workload differs in some aspects from the test data input of functional tests. Performance workload must represent multiple user inputs, not just one. The generation of performance workload may require dedicated tools or hardware. Additionally, the system under test should be free of functional defects that could impact the results of the performance tests.

Jiang and Hassan [49] argue that performance testing is trying to uncover load-related problems. Based on this observation, Jiang and Hassan [49] present two *workload design* strategies:

1. **Designing realistic loads:** The idea behind this strategy is to generate a workload that resembles realistic system usage. The target is to ensure that the system functions correctly in the field. If the system can handle a realistic workload without any functional or non-functional issues, it passes the performance test.

2. **Designing fault-inducing loads:** This strategy is actively trying to provoke functional and non-functional issues by generating a workload that is likely to cause issues.

Bath et al. [11] provide a list of *workload generation* approaches:

- Load Generation via the User Interface
- Load Generation using Crowds
- Load Generation via the Application Programming Interface
- Load Generation using Captured Communication Protocols

The above strategies are presented in the context of web applications but can also be transferred to network applications.

Jiang and Hassan [49] provide a general list of *workload generation* approaches:

- **Live-user based executions:** A load performance test examines the behaviour of a system used by multiple users. Employing human testers to generate a workload is, therefore, an intuitive solution. An advantage to this approach is that human testers can provide feedback about the system.
- **Driver based executions:** This approach overcomes the scalability issues of the live-user generation by generating the workload automatically. However, load drivers need setup and configuration. Tracking the behaviour of the system can be challenging too.
- **Emulation based executions:** The previous two approaches required a fully functional test system in a production-like environment. The emulation-based approach is conducted on a special platform that can emulate parts of the system. This allows for performance tests of parts of the system throughout the development lifecycle, before the system is completely ready.

6.5 Testing Environment

Yorkston [101] provides a general concept of a performance test environment, as presented in Figure 6.1.

A performance testing environment generally consists of three parts:

- **Test Controller:** The performance test controller executes the performance tests. It triggers the Load Generators at the beginning of a test and collects testing logs from the Load Generators and the system under test afterwards.

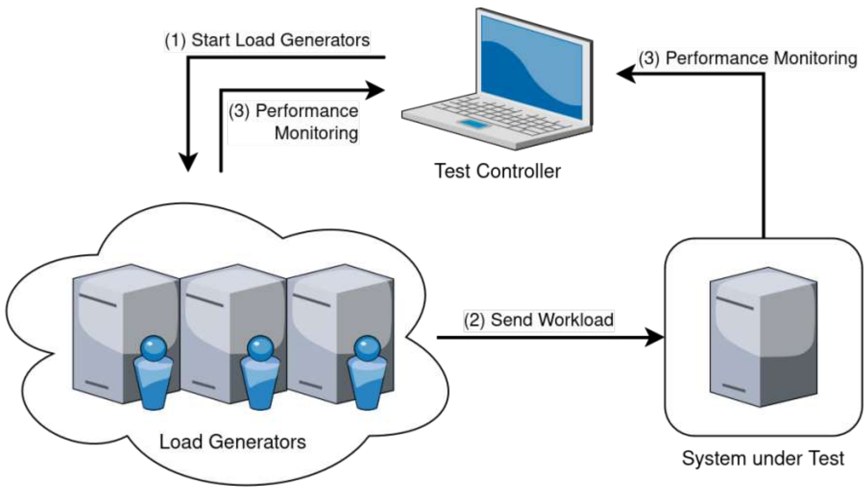

(1) Start Load Generators
(3) Performance Monitoring
Test Controller
(3) Performance Monitoring
(2) Send Workload
Load Generators
System under Test

Figure 6.1: The general concept of a performance testing environment (based on [101]).

- **Load Generators:** The Load Generators are one or more machines that generate the workload as requested by the Test Controller and send it to the system under test. The Load Generators can act as one or more users.
- **System under test:** The system under test reacts to the workload it receives from the Load Generators. Performance metrics such as response time and resource utilisation are captured during the test.

In general, a performance test contains the following steps:

1. **Start Load Generators:** The Test Controller contacts the Load Generators and provides configuration details for the desired workload.
2. **Send Workload:** The Load Generators start sending out load to the system under test according to the configuration provided by the Test Controller.
3. **Performance Monitoring:** The performance monitoring logs are collected at the Test Controller for further processing and analysis.

CHAPTER 7

Performance Measurement of WireGuard with HSM

The focus of this chapter is to present the results of the performance measurements of a WireGuard VPN with and without an extended HSM. Section 7.1 will detail how the performance tests were designed. Section 7.2 will discuss the considered performance metrics, and Section 7.3 will present how the performance metrics were monitored and logged while Section 7.4 will discuss how the logged metrics were visualised. Finally, Section 7.5 will present the test results.

7.1 Performance Test Planning

This section will provide a general outline of the carried-out performance tests. Section 7.1.1 will discuss the test targets, and Section 7.1.2 will delineate the used test setup. Section 7.1.3 will present the test plans, and Section 7.1.4 will show which test inputs were used for the test plans.

7.1.1 Test Targets

The following test targets were considered during the performance tests:

- **WireGuard Kernel:**[1] The Linux kernel implementation of the WireGuard protocol. This test target will serve as the baseline for all other test targets as it is expected to provide the best performance. The WireGuard kernel version was acquired from the official package repository of the operating system of the host system in version `1.0.20210914-1ubuntu2`.

[1]https://github.com/WireGuard/wireguard-linux

- **WireGuard Go:**[2] A WireGuard protocol implementation in the Go programming language. This test target will be of interest in two ways, on the one hand in comparison to the kernel version and on the other hand to the Go version using an HSM. WireGuard Go was acquired from the official WireGuard GitHub repository master branch in commit version `12269c2` and built manually.
- **WireGuard Go with HSM:** An adaptation of the WireGuard Go implementation to allow for storing a private key in an HSM. Test plans were executed for two different HSM. See Section 4.5 for details on the HSMs under test.

7.1.2 Test Environment

Following the observation discussed in Section 6.2 that the test environment should resemble the production environment as closely as possible this section will present the test setup that was used. The test setup for the performance tests consisted of these parts:

1. Server Host
2. Load Generator Host
3. HSM with KMS
4. Test Controller

In general, during each test plan execution, the Test Controller would first start monitoring scripts on the Server Host. Next, the Test Controller would tell the Load Generator Host which test to start. The Load Generator would then in turn start generating the requested load and send it to the Server Host.

The Server Host would contact the KMS and HSM during the test when necessary (see Section 5.4). Once the test execution was completed, the Test Controller would collect monitoring data from the Server Host and test reports from the Load Generator Host.

While the Load Generator Host was located as a virtual container in Austria, the Server Host, as well as the HSM and KMS, were located in Germany. The Test Controller was a laptop located in Austria. A performance test, where the Load Generator Host was also deployed in Germany, close to the KMS and HSM, showed similar results and mostly differed by an expected RTT decrease. The test results presented in this thesis refer to the setup where the Load Generator Host is located in Austria.

The RTT (see Section 3.2.6) between the Load Generator Host and the Server Host was measured to be around 18ms and the RTT between the Server Host and the KMS to be around 0.5ms. Figure 7.1 presents the general test setup that was used during the WireGuard performance testing.

[2]https://github.com/WireGuard/wireguard-go

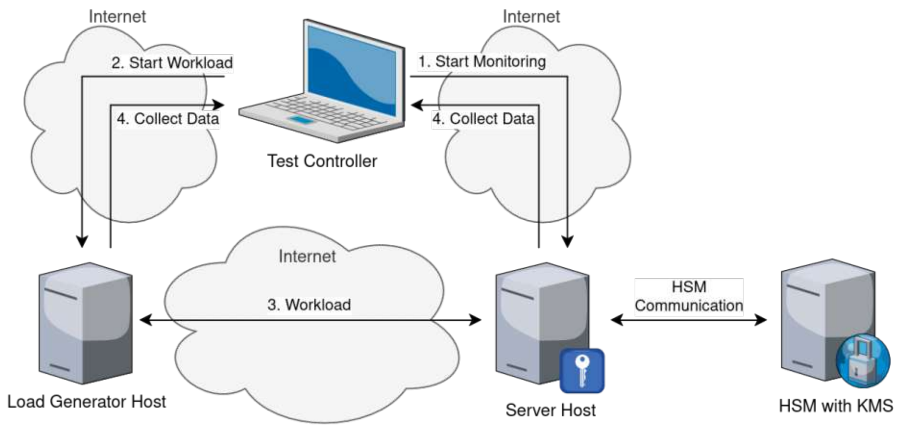


Figure 7.1: General test setup for the WireGuard performance tests.

Server Host: This host was used to run all tested versions of WireGuard. A script was used to generate the necessary private- and public-key pairs for the peers and write them to configuration files. The configuration files for the peers were collected by the Test Controller and transferred to the Load Generator Host. The peers were configured using the pre-shared key option and without the persistent keepalive option (see Section 5.4.1). These options were enabled and disabled respectively based on the requirements of the production system. The same configuration files were used for all tests. Another script that was based on the `wg-quick` [28] tool by WireGuard, was used to set up WireGuard on the Server Host. The hardware resources of the Server Host were chosen to be similar to that of the production system. Equally, the Server Host operating system was set up similarly to that of the production system, installing the same software to operate and monitor the host as in the production environment. Details about the hardware resources and the operating system of the server host can be found in Table 7.1.

Load Generator Host: This host was used to generate the test inputs for the test plans discussed in Section 7.1.3. To ensure the repeatability of tests, the input generation was done inside containers on the host. Details about the hardware resources and the operating system of the load generator host can be found in Table 7.1.

HSM with KMS: This system manages the generation and storage of cryptographic keys. The KMS provides an abstraction layer of the HSM. Systems such as the WireGuard server which need to access the HSM do not access it directly but through an interface provided by the KMS. During the performance tests, the KMS and HSM were considered as a black box and one related part.

Test Controller: This host was used to orchestrate the execution of tests. It was used to start monitoring on the Server Host, trigger the workload generation on the Load

Server Host		
CPU	Memory	Operating System
12 cores, 2744.844 MHz	15 GiB Memory, 1 GiB Swap	Ubuntu 22.04
Load Generator Host		
CPU	Memory	Operating System
8 cores, 2744.844 MHz	7 GiB Memory, 1 GiB Swap	Ubuntu 22.04

Table 7.1: Hardware resources of the test setup.

Generator Host, and collect the results of the tests from both hosts once the test was finished.

7.1.3 Test Suites

As discussed in Section 3.4.3, the first steps in performance testing are the planning and designing of the test plans. Based on the performance testing types discussed in Section 6.2 the following test plans were devised:

Latency Test Suite: This test suite aimed to establish a baseline of the performance of the test targets, allowing a comparison afterwards. The baseline was established by measuring the RTT for single and concurrent WireGuard handshake requests.

As discussed in Section 5.4.2, a handshake request is the first message from a peer to a server to initiate an encrypted communication. No communication can take place until the handshake has been carried out. Additionally, the performance of the HSM directly impacts the RTT of WireGuard handshakes since the DHKE is carried out inside the HSM (see Section 5.4.2). Therefore, the latency of the handshake request affects all subsequent messages. Of particular interest was also the concurrent processing of handshake requests by multiple peers. The here presented test plans aim to provide a baseline of the test target's performance on response times for handshake requests.

The aim was to carry out measurements for 1, 10, 25, 50, 100, 200, 500, 1000, 2000, 4000 and up to 8000 concurrent peers. By starting the tests with only one peer and then roughly doubling the number of peers with each additional test plan, a baseline of the test target's latency and a thorough overview of the test target's behaviour under load can be obtained. The maximum number of 8000 peers was the desired test target's maximum number of peers. The tests were stopped as soon as it was clear that the test system was at capacity.

As discussed in Section 5.4.3, the rekey timeout of WireGuard is set to five seconds. By setting the delay between each handshake request to 10 seconds, the test target has more time to recover between each request than in a real-world scenario.

The number of requests per peer for each test plan needs to be large enough to allow for statistical calculations but low enough such that the test plans can be carried out in a reasonable time frame. With a delay of 10 seconds between each handshake initiation, choosing a total of 100 requests per peer results in a runtime of almost 17 minutes, thus requiring almost four hours to execute all test plans.

Table 7.2 provides an overview of the devised test plans. Besides the test plans discussed, additional plans were used during the creation of this thesis, with a different number of peers, number of requests, or delay if necessary to get a more detailed picture of the test target.

Throughput Test Suite: This test suite aimed to determine the throughput the test targets can provide. Since the encryption and decryption of packets on the server are causing a workload for the server, it is necessary to determine how much throughput the provided resources can provide. While the expected bandwidth could simply be calculated by multiplying the expected number of peers with an upper bound of expected throughput per peer, the resulting load on the test target is unknown. Measurements were carried out for 1, 10, 20, and 50 concurrent peers which were all sending as much traffic as possible at the same time. As discussed in Section 5.4.3, rekeying of the encrypted communication is carried out every 120 seconds. The throughput test plans were carried out for 180 seconds to allow for rekeying to occur once during testing. Table 7.3 provides an overview of the used test plans.

Load Test Suite: This test suite aimed to measure the maximum amount of peers the test target can serve reliably. These test plans are defined by the total number of peers and the total throughput they create.

WireGuard peers generate load on the HSM by sending handshake initiations (see Section 5.4). A peer sends handshake initiations initially when establishing a connection and then at least every 120 seconds to maintain the connection. This means that a test target can only support as many peers as the underlying HSM can process requests in this period. Therefore, the highest load on the HSM would be created by spreading out the total number of peers over this period.

Peers were started consecutively in batches with a delay between each batch. For each batch, the batch size defines the number of peers that are started in parallel. The runtime defines how long each peer is running. All tests were run for more than 120 seconds to allow for rekeying to occur (see Section 5.4.3).

The initial load test for a test target was defined based on latency test results. Subsequent tests also took the previous load tests into account during their design.

A good starting position for a load test would be a latency test run whose results had no cookie responses or timeouts. This would suggest that the test target was able to handle the concurrent load (see Section 7.2). The test result should have a low maximal RTT, compared to other test runs, so that it is possible to start as many batches as possible. Additionally, the range of the latency test result should be small, as this would suggest that the HSM could provide reliable results.

Plan Name	Peers	Requests per Peer	Delay
1p100c10s	1	100	10
10p100c10s	10	100	10
25p100c10s	25	100	10
50p100c10s	50	100	10
100p100c10s	100	100	10
200p100c10s	200	100	10
500p100c10s	500	100	10
1000p100c10s	1000	100	10
2000p100c10s	2000	100	10
4000p100c10s	4000	100	10
8000p100c10s	8000	100	10

Table 7.2: List of latency test plans used for measuring the baseline response time of the test target.

Plan Name	Peers	Runtime
1p180s	1	180
10p180s	10	180
25p180s	20	180
50p180s	50	180

Table 7.3: List of throughput test plans used for measuring the baseline capacity of the test target.

Plan Name	Number of Batches	Batch Size	Batch Delay	Bandwidth per Peer	Runtime
70p10x7b16s50Kbs400s	7	10	16	50Kbs	400
600p15x40b3s1000Bs400s	40	15	3	1000Bs	400

Table 7.4: Exemplary list of load test plans used for measuring the maximum amount of peers for the test target.

The number of peers per batch was set to the number of peers of such a latency test result. The delay between each peer batch was initially chosen such that it is slightly larger than the maximum observed RTT of the identified latency test.

The throughput per peer was typically set to be 1000 bytes per second, as this was the expected load on the production system. Based on the results of the initial test, the number of peers per batch, the delay between batches, or both were changed to increase the total number of peers. The total number of peers would be increased until the test target shows clear signs of performance issues. Table 7.4 provides some examples of used test plans.

7.1.4 Test Inputs

The generation of workload is a central point in performance testing. Each of the test plans described in Section 7.1.3 needs a specific workload which will be discussed in this section. Based on the workload generation strategies discussed in Section 6.4 the following test inputs were used for performance testing.

The Latency Test Data: The latency test suite uses WireGuard handshake requests as input to determine the response time for a handshake. A Go script was created that could create and send handshake requests for 8000 individual peers within 400ms. The script uses the generated peer configuration files from the Server Host, which contain all the necessary information to create a valid handshake, as discussed in Section 5.4.2. By taking the number of peers to simulate, the number of handshakes to send per peer, and the time to wait between each request as an argument, the script would then send out concurrent handshake requests for each peer, measure the time for a response to arrive, and write the results to a file. Section 7.3 will discuss this report in more detail.

The Throughput Test Data: The throughput test suite needs to generate as much network traffic as possible to determine the throughput (see Section 3.2.6) of the test target. The `iperf` [66] tool can measure the maximum achievable throughput of a network. To this end, `iperf` uses a client instance on one host and a server instance on another host. By generating as much traffic as computationally possible on the client instance and sending it over the network to the server instance it is possible to measure the data rate of a network. Both, the client and server, generate reports containing information about the test. Section 7.3 will discuss the `iperf` reports in more detail. Because the `iperf` tool can be configured to use a VPN connection, it is a prime option to test the throughput of WireGuard, like it has been done by Donenfeld [29]. The general test setup consists of an `iperf` client instance on the Load Generator Host and an `iperf` server instance on the Server Host. By starting multiple instances of clients and servers in parallel, each using a different WireGuard peer configuration (see Section 5.4), it is possible to simulate multiple peers.

The Load Test Data: The load test suite needs to simulate any given number of peers, ranging from one to possibly thousands. Furthermore, it must be possible to start and stop peers in a defined way to create the highest possible load. A Go script was created, based on the WireGuard Go code, that could simulate WireGuard peers to a WireGuard server. To this end, the script would send out handshake initiations, read the received handshake response, and send encrypted data using the derived key (see Section 5.4). The script would measure the RTT for handshakes and write the results to a file. Section 7.3 will discuss this report in more detail. Just like an actual WireGuard peer, the script would send another handshake initiation after five seconds in case it did not receive a handshake response. In case of a cookie response, it could generate the correct handshake initiations afterwards, similar to an actual WireGuard peer.

7.2 Performance Metrics

Choosing adequate performance metrics to determine the performance of a test target is a crucial part of performance testing [11]. As discussed in Section 4, access to an HSM is restricted which limits the possibilities of collecting performance metrics severely. In this thesis, the Server Host performance is observed in detail. If the test target shows signs of performance problems, but the Server Host performance logs are unremarkable, it can be concluded that the HSM had performance problems. Based on the performance metrics discussed in Section 6.3 the following metrics were determined to be meaningful for WireGuard with HSM performance testing.

1. **CPU usage:** Monitoring the CPU usage on the Server Host was of critical importance to ensure that the Server Host has enough available resources to serve the presented traffic.

2. **Load average:** Similar to the CPU usage, the load average of the Server Host was of interest to determine if the Server Host has enough resources.

3. **Memory usage:** Should the Server Host run out of available memory it can no longer operate. It is, therefore, necessary to know how much memory the Server Host is using under load. During the performance test in this thesis, it was observed that the Server Host uses very little memory and never reaches the limits of the available memory. Therefore, it was decided not to graphically display the memory usage (see Section 7.4).

4. **Throughput:** Measuring the throughput of the test target is essential to determine if the traffic generated by the peers is arriving at its target. For each load test plan, there is an expected data throughput based on the number of peers and how much data each peer is sending each second. A mismatch between the measured and the expected throughput could point to lost network packets which would result in a performance loss for affected peers.

5. **Latency:** As discussed in Section 5.4, a WireGuard peer can not send encrypted packets until the first handshake is complete. The RTT of the first handshake affects the throughput to a large degree and is, therefore, of utmost importance as a performance metric. Subsequent handshakes are not as critical for the performance because WireGuard tries to refresh the encryption key 60 seconds before it is invalidated (see Section 5.4). However, if WireGuard would not be able to refresh the encryption key within this period, it would not be able to transmit any more data until the key has been refreshed.

7.3 Performance Monitoring

In order to monitor some of the metrics discussed in Section 7.2 a system monitor written in Python using the `psutil` library was used. This system monitor was logging the CPU

Timestamp	CPU usage	CPU load average	Memory usage	Top process	Received Bytes	Packets dropped
2024-03-29 13:05:00	0.3%	(0.134765625, 0.0546875, 0.0087890625)	10.8%	systemd (0.0% CPU)	21018684280	0
2024-03-29 13:05:01	0.5%	(0.134765625, 0.0546875, 0.0087890625)	10.8%	systemd (0.0% CPU)	21018684280	0
2024-03-29 13:05:02	7.5%	(0.134765625, 0.0546875, 0.0087890625)	11.2%	puppet (103.8% CPU)	21018684280	0
2024-03-29 13:05:03	8.6%	(0.134765625, 0.0546875, 0.0087890625)	11.3%	puppet (17.1% CPU)	21018684280	0

Table 7.5: An example output of a system-monitor output file.

```
Server listening on UDP port 5001 with pid 2338714
Binding to local address 10.1.104.222
Read buffer size: 1.44 KByte (Dist bin width= 183 Byte)
UDP buffer size:  208 KByte (default)
[  1] local 10.1.104.222%ens3 port 5001 connected with 10.13.0.2 port 47849 (sock=3) (
    peer 2.1.9) on 2024-03-29 09:25:32 (CET)
[ ID] Interval            Transfer     Bandwidth        Jitter   Lost/Total  Latency
    avg/min/max/stdev PPS NetPwr
[  1] 0.0000-1.0000 sec   15.8 KBytes   129 Kbits/sec   3.671 ms 0/11 (0%)
    9.508/-0.081/104.551/45.681 ms 12 pps  2
[  1] 1.0000-2.0000 sec   12.9 KBytes   106 Kbits/sec   2.072 ms 0/9 (0%)
    -0.005/-0.046/0.095/0.042 ms 9 pps -2835
[  1] 2.0000-3.0000 sec   12.9 KBytes   106 Kbits/sec   1.193 ms 0/9 (0%)
    -0.008/-0.070/0.193/0.077 ms 9 pps -1726
...
[  1] 0.0000-400.0112 sec  4.89 MBytes   102 Kbits/sec   0.749 ms 0/3486 (0%)
    0.382/-0.113/104.551/2.883 ms 9 pps 34
```

Figure 7.2: An example output of an `iperf` server report file.

usage, load average, memory usage, the top process as well as the number of received bytes on the Ethernet interface on a per-second basis. The reason for logging the top process was to identify issues with the operating system setup of the Server Host, since the host had more than just a minimal WireGuard installation but, as discussed in Section 7.1.2, also had software to operate the host. Table 7.5 shows an example output file of the system monitor used to monitor the performance of the WireGuard server host.

The `iperf` tool, used to simulate the network traffic of peers as discussed in Section 7.1.4, generates a detailed second-by-second report about transferred bytes and bandwidth. While the client reports also contain write errors and the packets per second transferred, the server report additionally reports on the jitter of the received traffic. To cover the throughput metric discussed in Section 7.2 the transferred bytes of the server report have been regarded. Figure 7.2 shows an example `iperf` report of an `iperf` server and Table 7.3 shows an example `iperf` report of an `iperf` client.

```
Client connecting to 10.1.104.222, UDP port 5001 with pid 38416 via peer1 (1 flows)
TOS set to 0x0 (Nagle on)
Sending 1470 byte datagrams, IPG target: 114843.75 us (kalman adjust)
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[  1] local 10.13.0.2%peer1 port 47849 connected with 10.1.104.222 port 5001 (sock=3)
    on 2024-03-29 09:25:31.724 (CET)
[ ID] Interval        Transfer     Bandwidth       Write/Err  PPS
[  1] 0.00-1.00 sec   14.4 KBytes  118 Kbits/sec   9/0        11 pps
[  1] 1.00-2.00 sec   12.9 KBytes  106 Kbits/sec   9/0         9 pps
[  1] 2.00-3.00 sec   12.9 KBytes  106 Kbits/sec   9/0         9 pps
...
[  1] 0.00-400.12 sec  4.89 MBytes  102 Kbits/sec  3485/0        9 pps
[  1] Sent 3487 datagrams
[  1] Server Report:
[ ID] Interval        Transfer     Bandwidth        Jitter   Lost/Total  Latency avg/
    min/max/stdev PPS  Rx/inP  NetPwr
[  1] 0.00-400.01 sec  4.89 MBytes  102 Kbits/sec   0.748 ms 0/3486 (0%)
    0.382/4294967.183/104.551/0.957 ms 8 pps 8/0(0) pkts 33.54
```

Figure 7.3: An example output of an `iperf` client report file.

Thread	Sent	Elapsed	Type	Passed
2401	2024-03-29 09:23:53	100025	2	true
2401	2024-03-29 09:23:58	95063	2	true
2401	2024-03-29 09:24:03	95034	2	true
2401	2024-03-29 09:24:08	87799	2	true

Table 7.6: An example report output of the Go script used for latency tests.

As discussed in Section 7.1.4, test inputs for the latency test plans were created using a Go script. The time measurements for the handshake responses were written to an output file, consisting of the peer number, a timestamp, the measured response time in milliseconds, the type of the response, and whether a response was captured at all. Table 7.6 shows an example Go script report. “Thread” marks the peer number, “Sent” marks the timestamp of the handshake request, “Elapsed” logs the RTT of the handshake, “Type” marks the type of handshake response (response or cookie) and “Passed” logs whether a response was captured at all. The report generated by the Go script for the load test input has the same structure.

7.4 Visualising Test Results

During performance testing, a large amount of data is generated which needs to be processed in order to make it usable, as discussed in Section 6.3. Foundational notions on statistics and their presentation are discussed in Section 3.5. Performance metrics reports were processed using Python and visualised using `matplotlib` [45].

Visualising Latency Test Results: When visualising the latency test results, it is of interest to present the distribution of the measured values as well as the types of responses. As presented in Figure 7.4, the plot for latency test results was split in half, where the upper half is concerned with the response time values and the lower half with the response types. To compare the test runs of the same test target and the test runs between different test targets, all latency test run results of a test target were visualised in a single plot.

As discussed in Section 3.5.2, two possible ways of plotting a distribution are to use a boxplot or violin plot. A blend of both plots was used to visualise the distribution of the response time values by overlaying the violin plot with a boxplot without outliers, to gain the advantages of both plots. Comparison of test runs with different numbers of peers is possible by plotting the results for each test run next to each other.

Visualisation of response types was done using pie charts. A separate pie chart was created for each test run to compare the amount of successful, cookie, and timeout results.

Visualising Throughput Test Results: When visualising the throughput test results, it is of interest to visualise the total amount of the throughput, the change of the throughput over time as well as the individual throughput of each simulated peer for test plans with more than one peer. As discussed in Section 3.5.2, a stacked bar plot is a way to achieve the desired goals.

Results of the throughput tests were presented for different numbers of simulated peers. The throughput test results are presented as a stacked bar plot. Each simulated peer was assigned a unique colour inside the stacked bars. Thus, in the case of a single simulated peer, the stacked bar plot consists only of a single colour bar. The height of each bar presents the achieved throughput for this peer at this point during the performance test. The height of the stacked bar presents the total throughput achieved. For each plot, the median total throughput was marked in the plot. See Figure 7.14 for an example of such a plot.

Visualising Load Test Results: The main goal when visualising the load test results was to determine whether peers would experience availability issues and if the provided hardware resources of the Server Host were sufficient to handle the load. To that end, the plot of a load test result is split into three parts. See Figure 7.7 for an example of such a plot.

The upper plot shows the CPU usage and load average as a time-series plot on the same X-axis because, as discussed in Section 6.3, CPU usage and load average are tied to each other.

The middle plot is visualising the total throughput during the test. The `iperf` reports created during testing were aggregated and plotted as a time-series plot. Additionally, received bytes of the network interface captured by the system monitor discussed in Section 7.3, were added on the same X-axis.

The lower plot is concerned with the visualisation of the measured response times to handshake initiations. Because it was of interest to visualise a general trend in response times and not response times for individual peers, the results were plotted as a time-series plot with single plots for each measurement. The type of the response was encoded by colour.

By plotting the latency response times as well as CPU usage and load average alongside the throughput results it is possible to determine the cause of a throughput interruption.

7.5 Test Results

This section presents the results of the performance tests of a WireGuard server. Sections 7.5.1 through 7.5.4 will present the results of the test suites discussed in Section 7.1.3.

7.5.1 WireGuard Kernel Test Results

As discussed in Section 7.1.1, an original and unaltered version of WireGuard for the Linux kernel was used to create a baseline of comparison for all other test targets. As discussed in Section 7.1.4, latency tests were performed for up to 4000 concurrent peers. The result of the latency tests for the Linux kernel version of WireGuard can be seen in Figure 7.4, see Section 7.4 for a general description of the plot.

The median value of response times over all test runs lies between 18.97ms and 23.18ms. The lowest median RTT was measured with 100 simultaneous peers and the highest RTT with 4000 simultaneous peers.

Of interest is the performance drop-off between 500 and 1000 peers. Up until 500 peers all handshake requests were answered successfully. Starting with 1000 peers some handshake requests were not answered within 10 seconds (timeout) and starting with 4000 peers some handshake requests were answered with cookie replies.

The number of timeouts during WireGuard kernel latency tests was as follows:

- 1000 peers: 85 of 100000 (0.085%.)
- 2000 peers: 7 of 200000 (0.0035%)
- 4000 peers: 1 of 400000 (0.00025%)

The number of cookie replies during WireGuard kernel latency tests:

- 4000 peers: 35644 of 400000 (8,911%)

7.5.2 WireGuard Go Test Results

As discussed in Section 7.1.1, an original and unaltered version of WireGuard written in Go was used to compare its performance with the Linux kernel version of WireGuard and with the altered version of WireGuard Go using an HSM. Comparing the latency

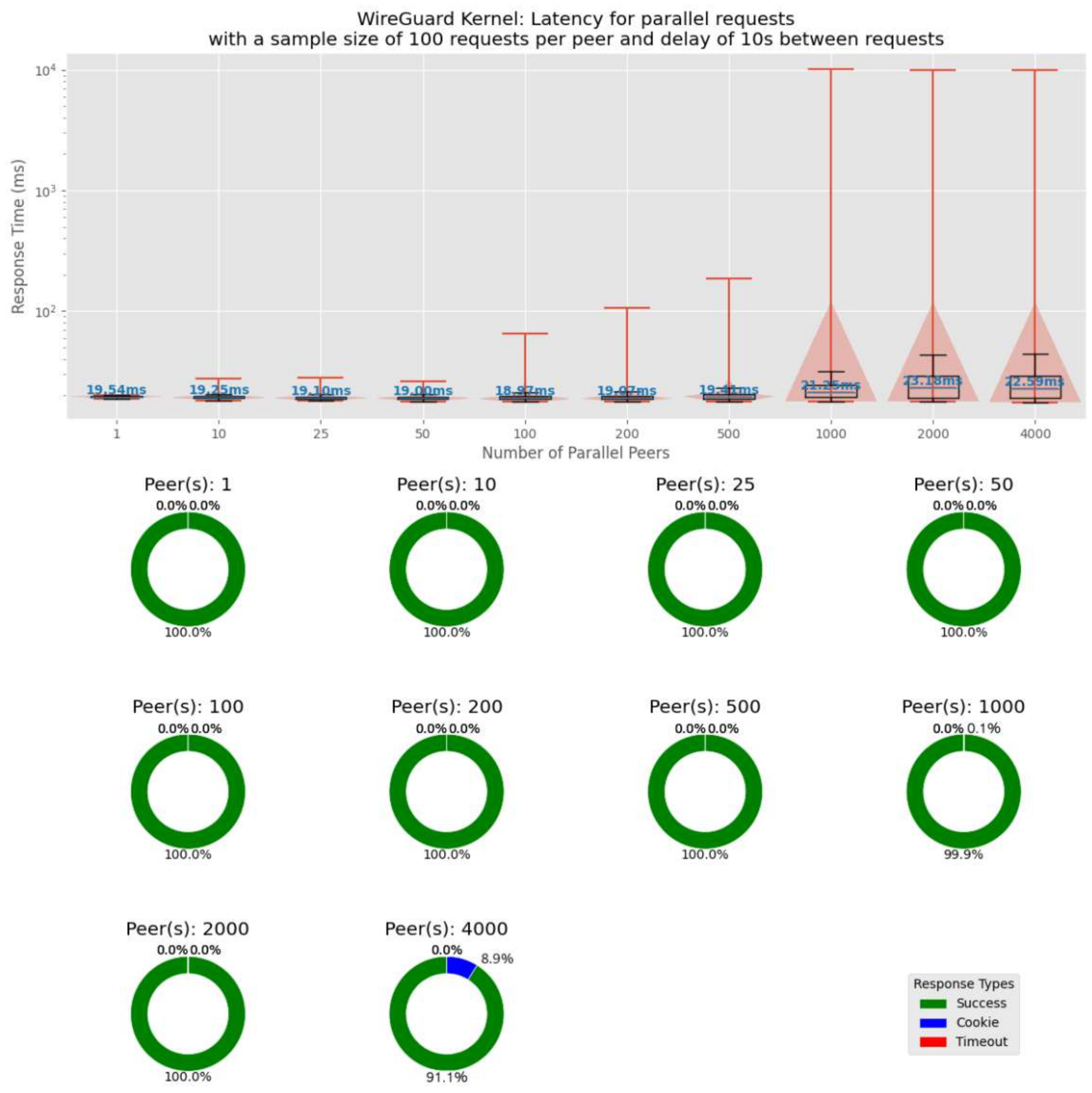


Figure 7.4: Comparison of individual latency test runs for the Linux kernel version of WireGuard.

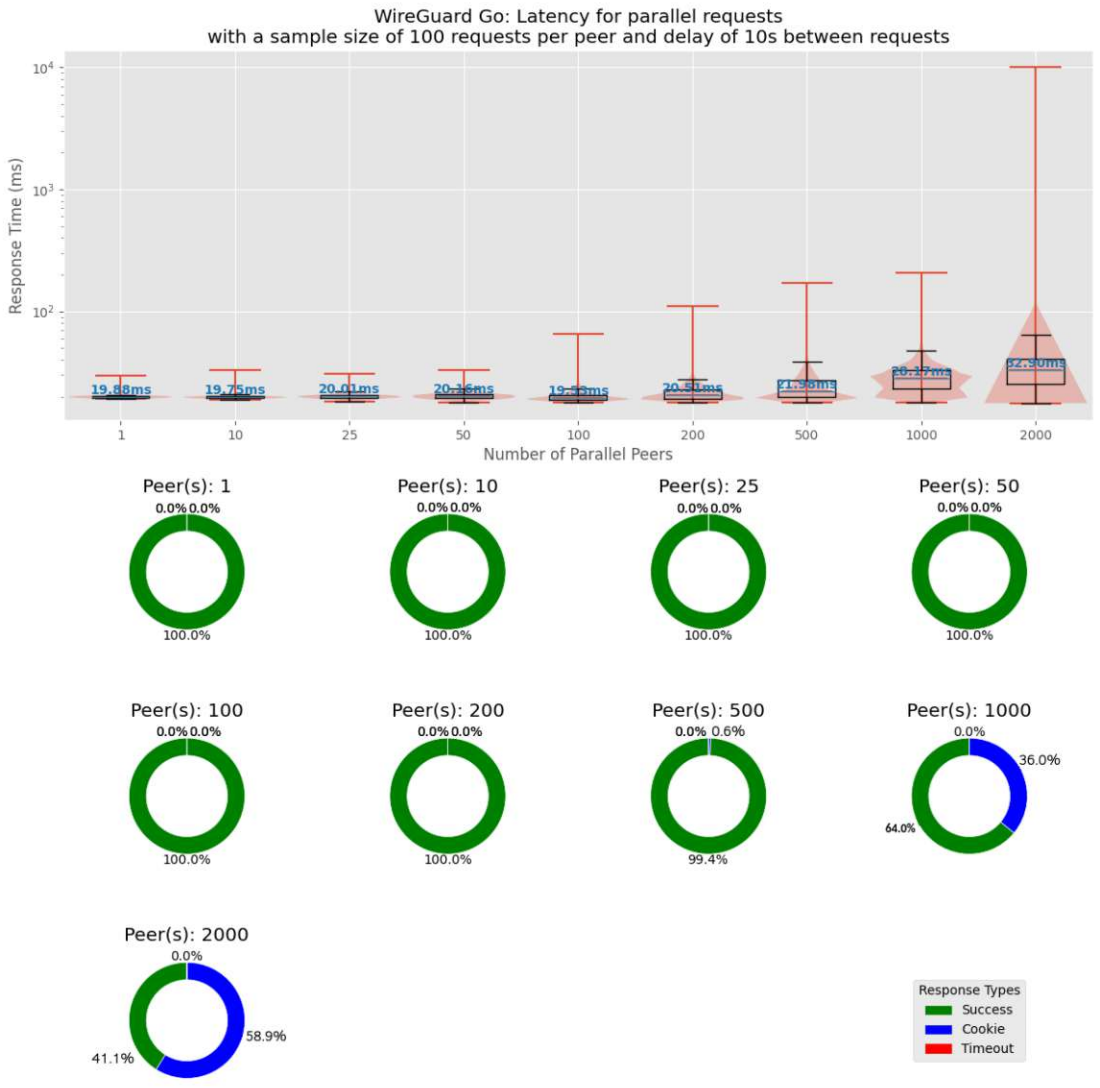


Figure 7.5: Comparison of individual latency test runs for WireGuard Go.

test results of WireGuard kernel version in Figure 7.4 with the test results of WireGuard Go version in Figure 7.5 it is apparent that the kernel version performs better than the Go version, even in the single peer test run.

The median value of response times over all test runs lies between 19.53ms and 32.9ms. The lowest median RTT was measured with 100 simultaneous peers and the highest RTT with 2000 simultaneous peers.

Of interest is the fact that during the WireGuard Go latency tests much fewer timeouts were recorded. No timeouts occurred during the 1000 peer test run. The first timeouts appeared during the 2000 peer test run with 5 out of 200000 (0.0025%) timeouts which

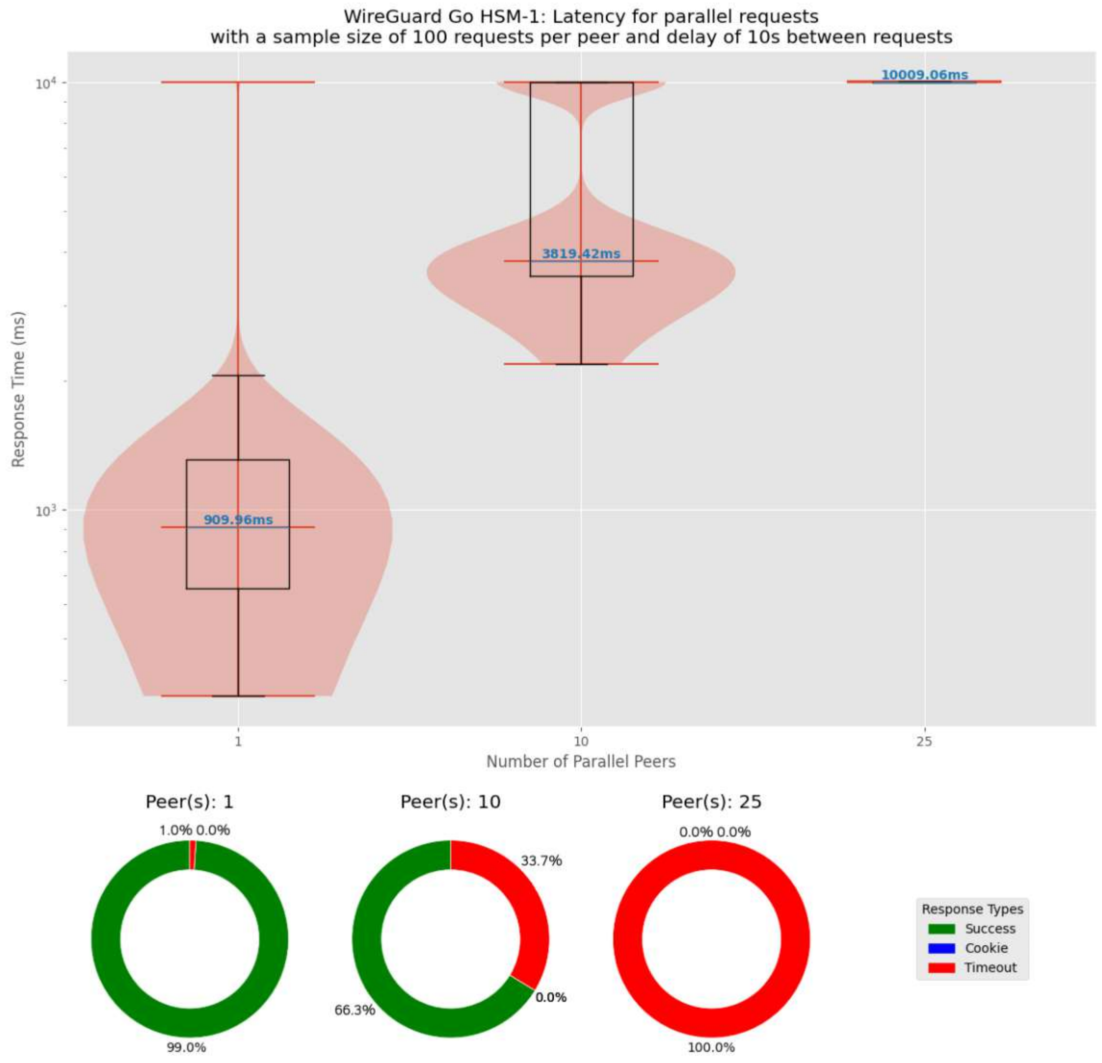


Figure 7.6: Comparison of individual delay test runs for WireGuard Go with HSM-1.

is similar to the 4000 peer WireGuard kernel test run.

7.5.3 WireGuard Go with HSM-1 Test Results

Latency tests with WireGuard Go together with the HSM-1 were carried out. Details about HSM-1 can be found in Section 4.5. The results of the latency tests for WireGuard Go with HSM-1 can be seen in Figure 7.6.

Compared to the latency test runs with the WireGuard kernel and Go version it is apparent that the test target using HSM-1 performed significantly worse. The single

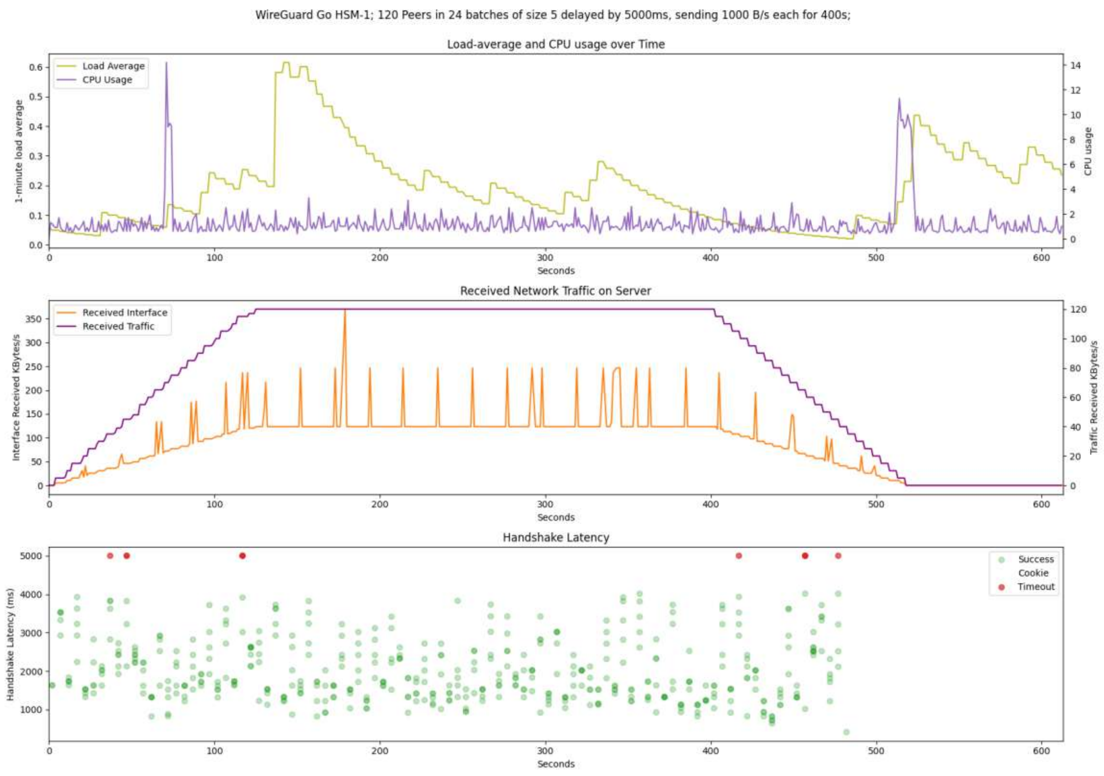


Figure 7.7: Load test for WireGuard Go with HSM-1, simulating 120 peers in total.

peer latency baseline test shows a median RTT of 909.96ms. There even occurred some timeouts during this test. During the 10 peer latency test run, the median RTT already rose to 3819.42ms. Although around 33% of all handshakes of this test run resulted in a timeout, WireGuard did not once answer with a cookie reply. This suggests that while WireGuard can handle the simultaneous load, the HSM seems to be having issues with it. During the 25 peer latency test run, none of the handshakes were answered within the given timeout.

As discussed in Section 7.1.3, based on the results of the latency tests the following initial load test was devised. Since the HSM was not able to handle 10 simultaneous peers, a batch size of five peers was chosen. A delay of five seconds between each batch was chosen because the violin plot of the 10 peer latency test run suggests that most successful answers were within that time. The resulting load test run simulated a total of 120 peers.

Although some timeouts were captured during the initial load test run, the overall result was positive. As can be seen in the top plot of Figure 7.7, resources on the Server Host were not nearly exhausted. Although the test run encountered two timeouts during the initial build-up of the load, as can be seen in the middle plot, the overall progress was satisfactory. The lower plot reveals that most handshake requests were answered between

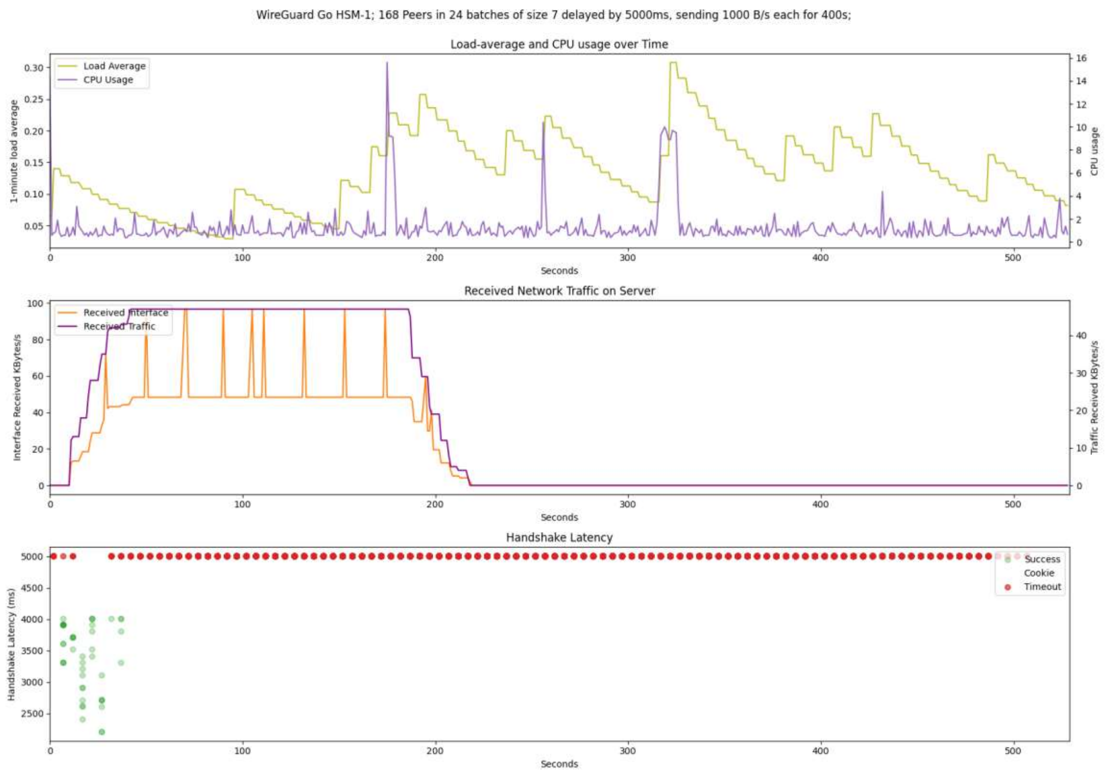


Figure 7.8: Load test for WireGuard Go with HSM-1, simulating 168 peers in total.

one and four seconds. This result aligns perfectly with the latency test results discussed earlier. An additional test run was carried out with one peer per batch and a delay of one second between each batch, based on the results of the single peer latency test results. The results were identical and did not suggest any advantages over the previous test run.

Although the initial load test was successful, the encountered timeouts suggest that the test target would not be able to handle more peers. Additional test runs were carried out trying to raise the total peer number. The results of one such run can be seen in Figure 7.8.

In contrast to the initial test run, the peers per batch were raised from five to seven, resulting in 168 total peers. Although some of the initial handshake requests were answered and some traffic was received at the Server Host, the test target was quickly overwhelmed and could not answer any more handshakes within the timeout. The result was that only a fraction of the peers were able to send traffic through the VPN, as can be seen in the middle plot of Figure 7.8. However, after 180 seconds the keys were discarded (see Section 5.4) and since the HSM was unable to answer any handshakes within the timeout, no more throughput was achieved. Because the Server Host did not experience significant CPU load, as can be seen in the top plot of Figure 7.8, it can be concluded that the HSM did reach its performance limit.

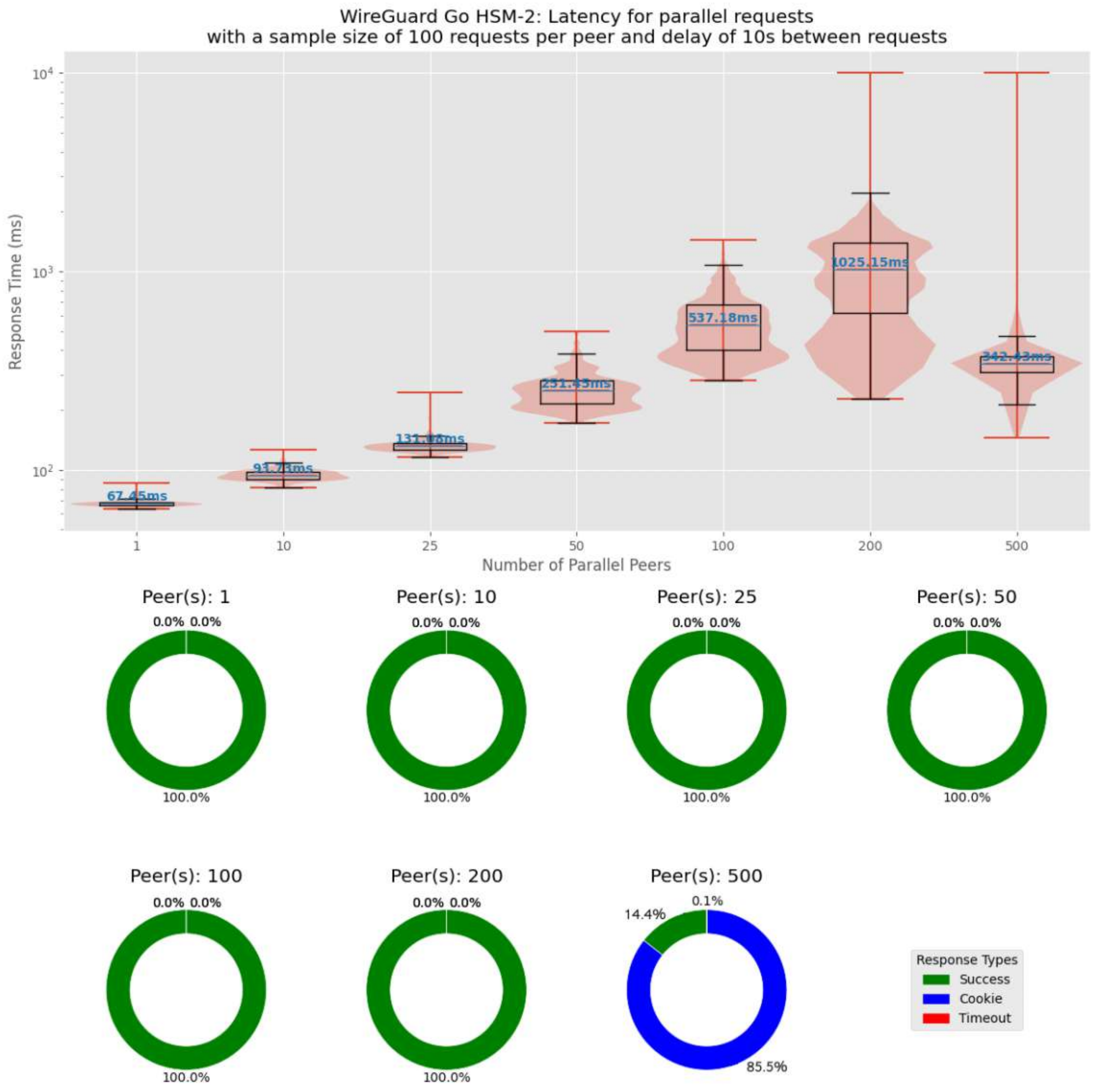


Figure 7.9: Comparison of individual latency test runs for WireGuard Go with HSM-2.

7.5.4 WireGuard Go with HSM-2 Test Results

Latency tests with WireGuard Go together with the HSM-2 were carried out. Details about HSM-2 can be found in Section 4.5. The results of the latency tests for WireGuard Go with HSM-2 are shown in Figure 7.9.

Compared to the latency test runs with the WireGuard kernel and Go version it is apparent that the test target using HSM-2 performed worse. However, compared to the test results with HSM-1 it is a clear improvement. In contrast to the test targets without an HSM, this test target shows the best performance for the single peer test run. The test target performance decreased with increasing peer number and the median RTT roughly doubled with double the peer number.

The first timeouts appeared during the 200 peer test run with 3 out of 20000 (0,015%) timeouts. The number of timeouts during WireGuard kernel latency tests:

- 200 peers: 3 of 20000 (0.015%)
- 500 peers: 41 of 50000 (0,082%)

The first cookie replies were recorded during the 500 peer test run. The number of cookie replies during WireGuard Go HSM-2 latency tests:

- 500 peers: 42761 of 50000 (85,522%)

As discussed in Section 7.1.3, based on the observations from the latency tests, load test runs were carried out. The starting point for the load tests was the 10 peer latency test run.

The initial load test involved initiating 10 peers every 150ms, culminating in a total of 8000 peers. Results of this load test are presented in Figure 7.10. Of interest are two occurrences where the handshake latency rose to around 4 seconds, presented in the bottom plot. However, the test target managed to get the latency down again without the use of cookie replies. The middle plot shows that the received network traffic was stable throughout the test run. CPU usage and load average are well within acceptable limits, as shown in the top plot. The result suggests that the test target could handle the load well.

The next load test involved initiating 10 peers every 120ms, culminating in a total of 10000 peers. Results of this load test are presented in Figure 7.11. Similar to the previous load test run, there is an occurrence where the handshake latency rose to around 4 seconds, presented in the bottom plot. In contrast to the previous load test run, the test target had to make use of cookie replies to manage the load. Nonetheless, the received network traffic presented in the middle plot does not reveal any issues with the throughput. The CPU usage peaked at around 35%, which is 5% more than in the previous load test run. Of concern is the spike in load average, around 180 seconds into the test run, presented in the top plot. A detailed review of the monitor log files suggests that the spike in the load average can be attributed to monitoring software installed on the system that is used in the production environment and not related to the test target. In any case, the spike in the load average only lasted for a short period and did not seem to influence the throughput. The result suggests that the test target could handle the load well.

The next load test involved initiating 10 peers every 100ms, culminating in a total of 12000 peers. Results of this load test are presented in Figure 7.12. Similar to the previous load test runs, there are occurrences where the handshake latency rose to around

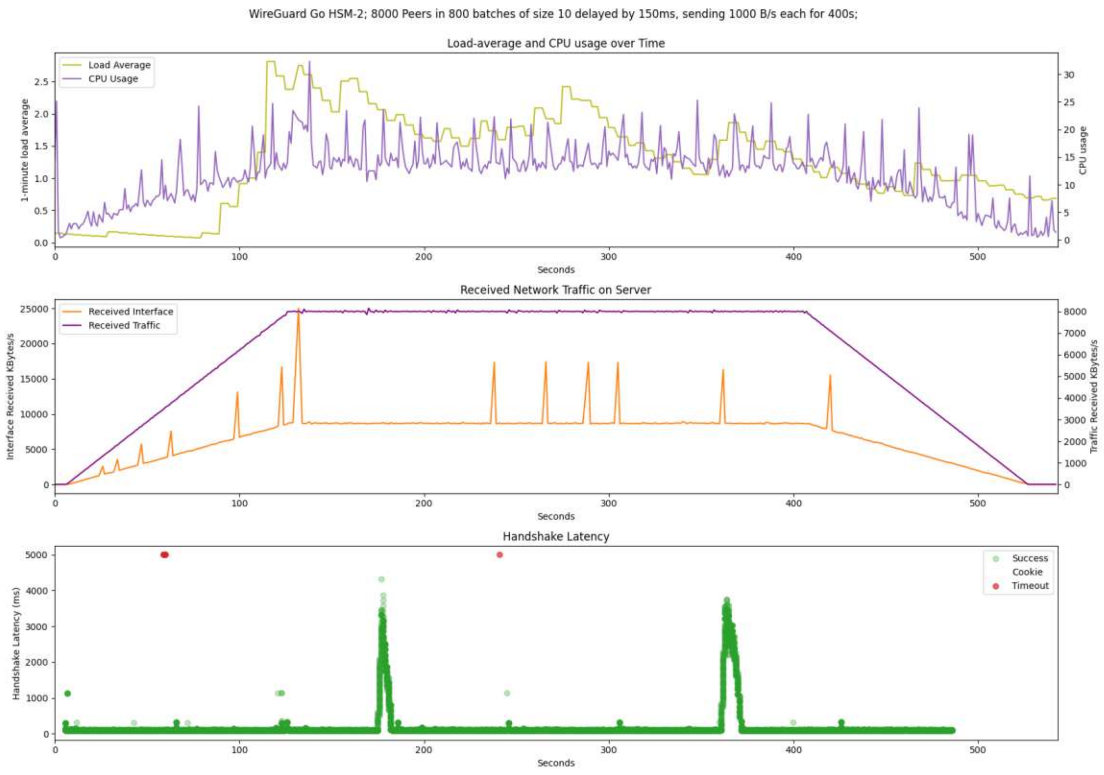


Figure 7.10: Load test for WireGuard Go with HSM-2. The test starts 10 new peers every 150ms for 120 seconds.

4 seconds, presented in the bottom plot. The test target had to make use of cookie replies to manage the load. While the test target was able to handle the load during the first occurrence, the test target seemed to struggle after the second occurrence which is noticeable by the constant appearance of cookie replies in the bottom plot. From around 350 seconds after the test started, the throughput decreased continuously suggesting that some clients were not able to send packets because of outdated keys (see Section 5.4.3). The CPU usage peaked at around 40%, which is again 5% more than in the previous load test run. Although the Server Host did experience some CPU load, as can be seen in the top plot of Figure 7.12, the Server Host was far from its limit. It can thus be concluded that the HSM did reach its performance limit.

Based on the results of the 200 peer latency test runs a final load test involved initiating 200 peers every 2000ms, culminating in a total of 12000 peers. Of interest was to evaluate how the test target would handle the longer delay paired with the larger peer batch. Results of this load test are presented in Figure 7.13. A quick review of this test run presents a noticeably worse performance compared to the previous test run. The result suggests that the test target could not handle the load and that the test target is better able to handle smaller peer batches in shorter delays.

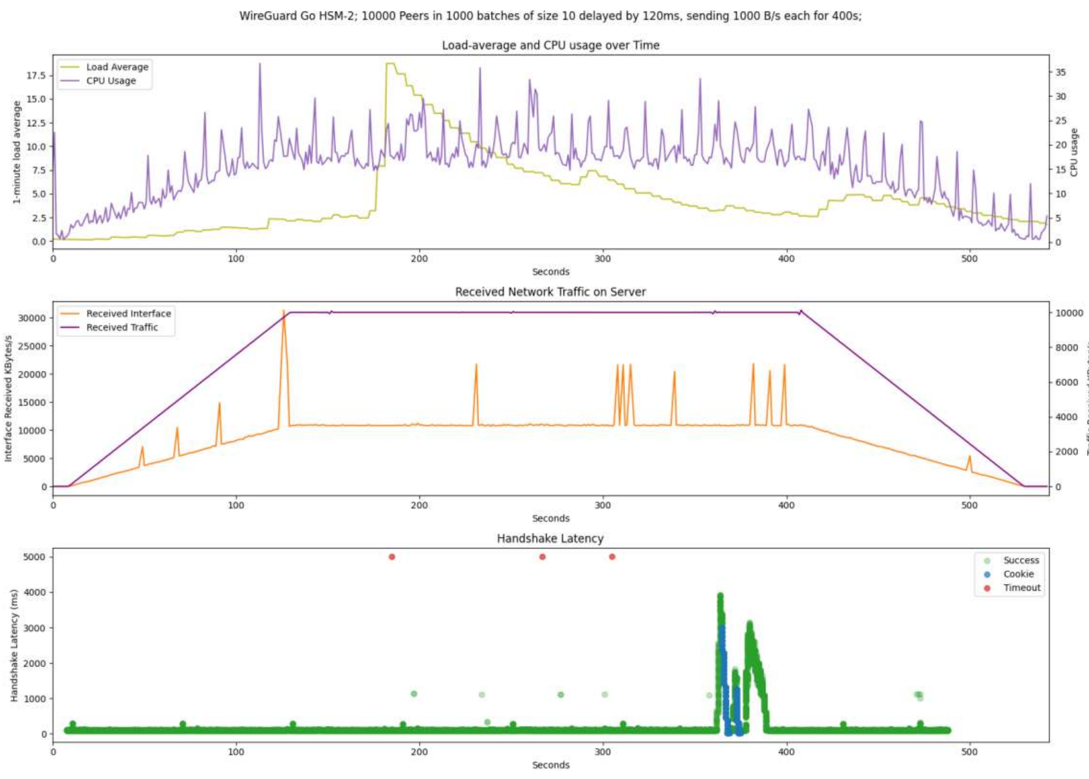


Figure 7.11: Load test for WireGuard Go with HSM-2. The test starts 10 new peers every 120ms for 120 seconds.

7.5.5 WireGuard Throughput Test Results

This section will discuss the results of the throughput test suite as presented in Section 7.1.3. See Section 7.4 for a description of the presented figures in this section.

The results of the throughput test runs for the WireGuard kernel version can be seen in Figure 7.14. It presents the results of test runs for 1, 10, 16, and 20 peers. Observing the CPU usage during the 1 peer throughput test run revealed that `iperf` was utilising only one CPU core but to 100%. This indicates that a single `iperf` instance was not enough to fully load the system. The 10 peer test run increased the throughput, in contrast to the 1 peer run, by 9% from 759.3Mbs to 828.3Mbs. The 16 peer test run increased the throughput, in contrast to the 1 peer run, by 11% from 759.3Mbs to 843.5Mbs. However, the 20 peer run showed a decrease in throughput in contrast to the 16 peer run by 0.3%. This suggests that using one instance of `iperf` for each available CPU core would produce the highest possible throughput. Because of this, 16 peer test runs were used to compare the throughput test results between the test targets. Although individual peers did experience some throughput fluctuations during test runs, the overall throughput was relatively stable during all runs that used multiple peers.

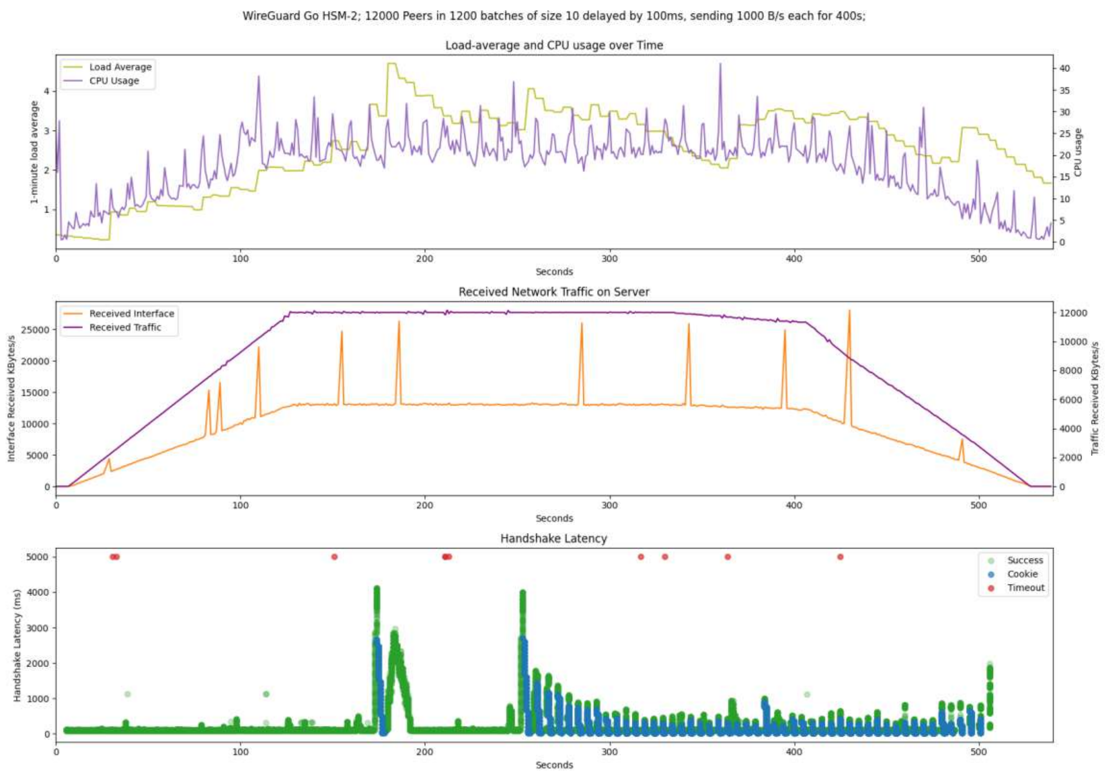


Figure 7.12: Load test for WireGuard Go with HSM-2. The test starts 10 new peers every 100ms for 120 seconds.

Figure 7.15 presents a comparison between the 16 peer throughput test runs for each test target. The WireGuard kernel version showed the highest throughput of 843.5Mbs. The WireGuard Go version showed a decrease in throughput, compared to the WireGuard kernel version, by 5.4% from 843.5Mbs to 800.15Mbs. The WireGuard Go version extended with HSM-2 showed a decrease in throughput, compared to the WireGuard Go version, by 3% from 800.15Mbs to 776.15Mbs.

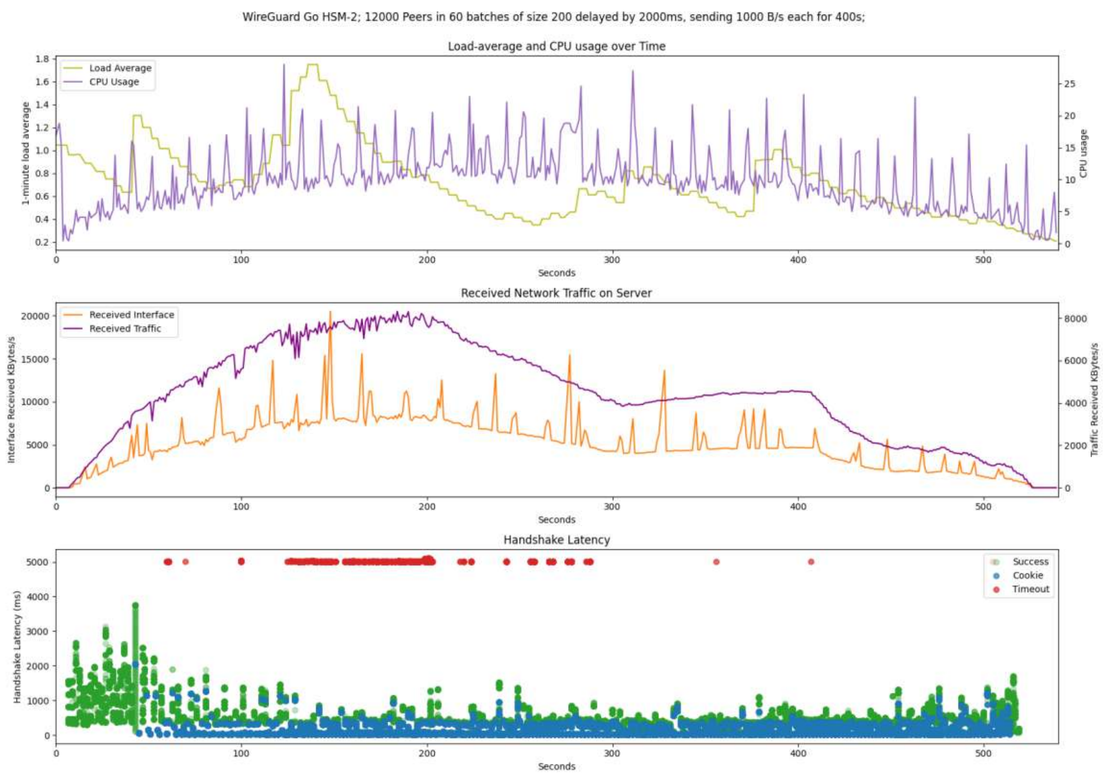


Figure 7.13: Load test for WireGuard Go with HSM-2. The test starts 200 new peers every 2000ms for 120 seconds.

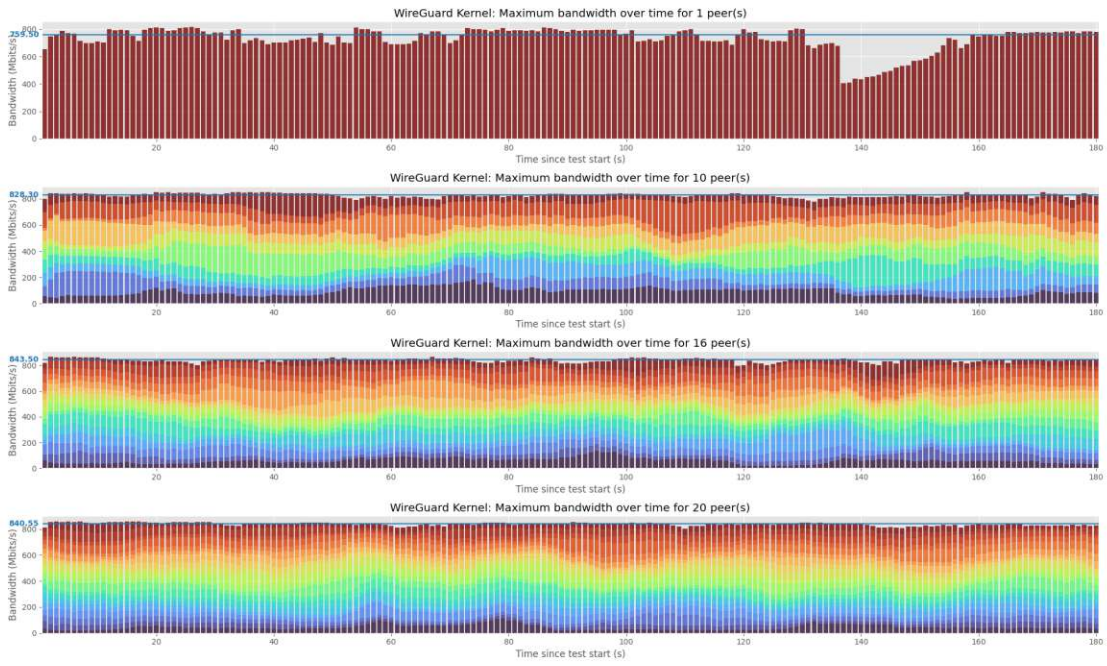


Figure 7.14: Throughput test results for the Linux kernel version of WireGuard.

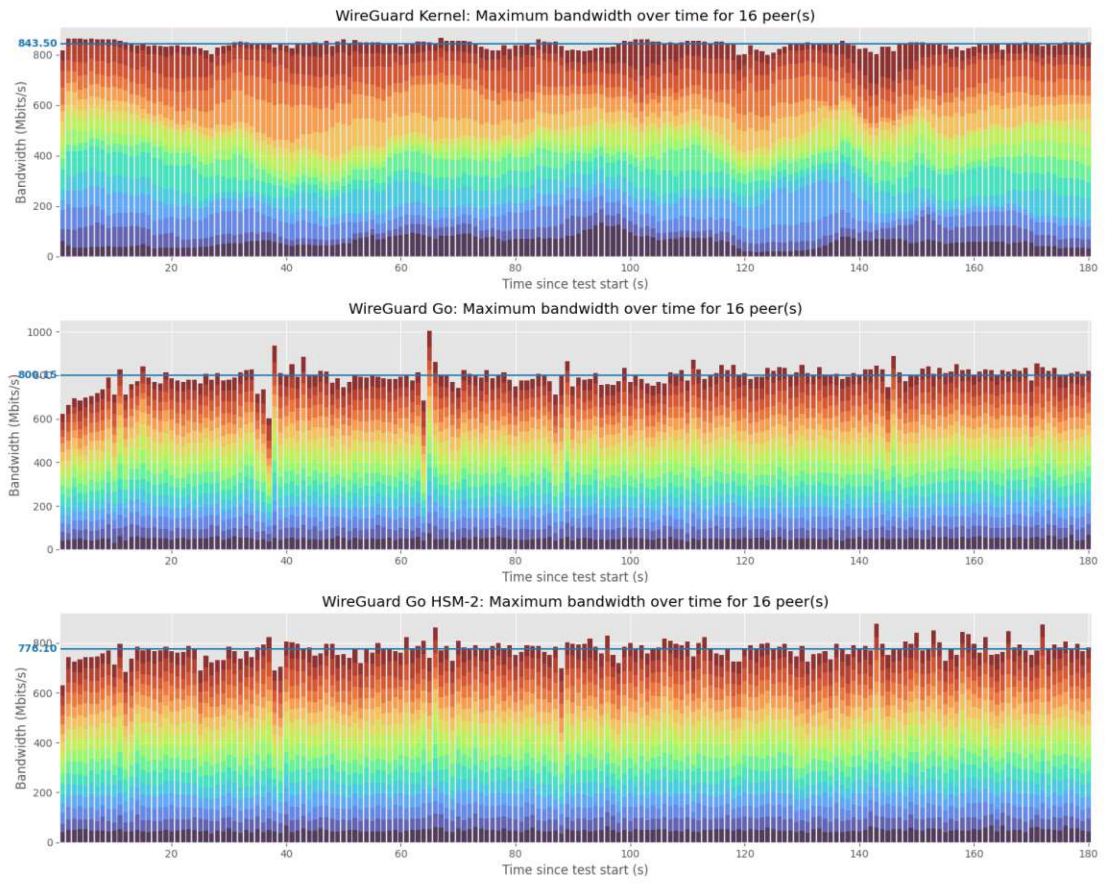


Figure 7.15: Comparing throughput test results between different test targets.

CHAPTER 8

Evaluation/Discussion

This chapter will first discuss the research questions presented in Section 1.3 and then go on to discuss the main hypothesis of this thesis.

RQ1. To what extent does secure access to remote services influence performance from an end user's perspective? Because an HSM is a dedicated device that is built to perform cryptographic operations it is expected to generally perform cryptographic operations faster than a general purpose server [51]. It could be expected that moving security-sensitive tasks, such as key management and cryptographic operations, inside an HSM could result in a performance increase. However, the results of this thesis show that this is not always the case. Because the WireGuard protocol is already very fast with its cryptographic operations (see Section 7.5.1), the introduction of an HSM is adding overhead that is decreasing performance.

Comparing the results of the tests without an HSM (see Section 7.5.2) and with an HSM (see Section 7.5.4) suggest that there is a performance decrease when using an HSM to secure a remote access. However, from an end user's perspective, this performance decrease might only become noticeable if the remote access in use is prone to latency. Otherwise, the performance decrease might go unnoticed, such as in the case of WireGuard because of the grace period allowed by the protocol (see Section 5.4.3).

RQ2. How can secure remote access to central resources be scaled? Based on literature research this thesis presented three different scaling methods in Section 4.4.

The scaling method proposed by Han et al. [43] is already indirectly used in the case of WireGuard. The WireGuard server stores the ephemeral keys itself and only contacts the HSM to carry out the DHKE to generate them, which involves the static secret keys (see Section 5.4.2).

Horizontal scaling (scaling out) by adding additional HSMs can be difficult. It requires load balancing between HSMs which can be difficult because it would require current performance data, such as CPU load and memory usage, of the running HSMs. HSMs typically do not provide an easy way to acquire such performance data. An example of a horizontal scaling solution given by Aref and Ouda [7] showed a decrease in performance.

Vertical scaling (scaling up) an HSM by increasing its resources is not possible because it would harm the integrity of the hardware and might even lead to its destruction because of its tamper resisting features. An HSM modified in this way would likely also lose its certification.

Instead, this thesis tested vertical scaling by replacing an HSM with a stronger model. The results showed an increase in concurrent users and a decrease in response time when using a more powerful HSM (compare Sections 4.5.1 and Section 4.5.2).

RQ3. Which characteristics are relevant to assess the suitability of secure cryptographic key storage in a high-performance environment? Table 8.1 presents a summary of the performance test results discussed in Section 7.5. The results of the performance test presented in Section 7.5.3 and Section 7.5.4 suggest that the performance of the test target depended largely on the performance of the HSM in use. The HSM-2 (22000 tps) has a stated performance roughly 32 times higher than the HSM-1 (680 tps), or in other terms, the HSM-1 has 3.09% of the performance of HSM-2 (see Section 4.5).

During the latency tests, the test target using HSM-2 displayed a latency 13.5 times smaller (67.45ms) than the test target using HSM-1 (909.96ms), or in other terms, the latency with HSM-2 was 7.4% of that of HSM-1. The HSM-1 has achieved a maximum stable user count of 120 (see Section 7.5.3), whereas the HSM-2 has achieved a maximum stable user count of 10000 (see Section 7.5.4). Expressed in percent, the test target using HSM-1 reached only 1.2% of the stable users reached with HSM-2.

Taken together, it can be concluded that the stated performance of an HSM roughly translates to the number of users the test target can handle. Although the actual performance of a secure remote access is difficult to predict using the performance data provided by manufacturers, it is possible to estimate the performance of other HSMs based on measured performance data and the stated performance.

Based on the answers to the research questions, the main hypothesis will now be discussed.

HSMs are suitable in securing the remote access in high-performance environments. Although the performance of WireGuard with HSM-2 is not as good as that of the WireGuard Go or kernel version, it was still possible to achieve a performance that could handle multiple thousand peers. The measured performance would be sufficient to be used in a production environment. Thus, the main hypothesis of this thesis is considered to be valid.

	WireGuard Kernel	WireGuard Go	WireGuard Go HSM-1	WireGuard Go HSM-2
Concurrent peers	Median RTT			
1	19,54ms	19,88ms	909,96ms	67,45ms
10	19,25ms	19,75ms	3819,42ms	93,73ms
25	19,10ms	20,01ms	>10000ms	131,08ms
50	19,00ms	20,16ms	-	231,45ms
100	18,97ms	19,53ms	-	537,18ms
200	19,07ms	20,51ms	-	1025,15ms
500	19,41ms	21,98ms	-	342,43ms
1000	21,25ms	28,17ms	-	-
2000	23,18ms	32,90ms	-	-
4000	22,59ms	-	-	-
	Median bandwidth for 16 concurrent peers			
	843,50Mbits/s	800,15Mbits/s	-	776,10Mbits/s
	Maximum stable user count			
	-	-	120	10000

Table 8.1: Summary of performance test results presented in Section 7.5.

CHAPTER 9

Conclusion and Future Work

HSMs are hardened physical computing devices, dedicated to performing cryptographic operations and for key management [51]. These devices are used in security-sensitive contexts, such as banking, insurance [86], and health [65]. However, frequent key operations can cause an HSM to become the bottleneck of a system [42]. This thesis addresses whether a remote access that moves cryptographic operations into an HSM in an environment that frequently performs cryptographic operations is performant enough in a high-performance environment. The author is not aware of any scientific work devoted to this topic.

To answer this question, the VPN protocol WireGuard was used as an example of a remote access application that frequently performs cryptographic operations. The WireGuard protocol is rotating its session key every 120 seconds. This key rotation causes a WireGuard server, handling multiple thousands of peers, to potentially calculate numerous DHKE per second. By moving this workload into an HSM this thesis aims to evaluate the suitability of on premise HSMs to operate in highly performance demanding environments.

Performance tests were carried out to determine the impact of extending a WireGuard server with an HSM. The most important performance metric in this context was determined to be the RTT of WireGuard handshakes as this latency is directly related to the computing performance of the HSM. The performance tests consisted of concurrent baseline testing and load testing. Baseline testing was used to measure the RTT of WireGuard handshakes for single and multiple peers. Based on these tests, load tests were formulated that simulated multiple concurrent peers.

A literature research on scaling methods revealed three solutions. This thesis showed that vertical scaling, by exchanging the HSM in use with a more powerful one, is a simple solution that increases the performance of the secure remote access. Aref and Ouda [7] showed that horizontal scaling is possible, however with a performance decrease.

This thesis tested two different HSMs. The results of the performance tests show a noticeable drop-off in peers that a WireGuard server extended with an HSM can serve, in contrast to a WireGuard server without an HSM. However, the results of the load tests revealed that the number of peers a WireGuard server can handle largely depends on the computing power of the used HSM. The more capable HSM of the two tested was able to achieve a performance that could be used in a real-world production environment.

Common HSM characteristics provided by manufacturers are given in transactions per seconds for a set of cryptographic operations. Characteristics are typically given for RSA and some elliptic curve cryptography. This provided performance characteristics roughly translates into actual performance for securing a remote access in a high performance environment. The results of this work indicate that the impact on performance depends in large part on the communication path between the test target and the HSM which negatively affects the overall performance of the system and must be kept as small as possible. It can be concluded that HSMs can provide cryptographic operations and key management in environments with high-performance requirements, but with some performance loss.

In the course of this thesis, various topics were identified for future work.

The performance tests revealed a performance drop-off between the WireGuard kernel version and the Go version. It would be interesting to see if the performance drop-off would be less significant if the kernel version of WireGuard were extended with an HSM. Additionally, versions of WireGuard using other programming languages such as Haskell might be of interest.

This thesis did consider the union of KMS and HSM as a black box. The author suspects that performance improvements can be achieved by examining the behaviour of those components during performance testing more closely. Additionally, the results of the HSM-1 performance tests suggest there might be some performance improvements possible in the source code of the HSM.

In the case of the WireGuard VPN the main load of cryptographic operations inside the HSM is caused by session handshakes. The main reason for this is that WireGuard rotates the session keys every 120 seconds. Future work could evaluate the effects of raising this rekeying interval.

The thesis found that the network latency between the Server Host and the KMS makes a significant contribution to the overall RTT and thus the performance of the system. Future work could focus on network architectures and technologies that could lower this latency.

During the test of HSM-1, the WireGuard server did not respond with cookie replies although it took the HSM a considerable while to answer its requests. The exact conditions under which a WireGuard server responds with a cookie reply are not discussed in the original paper by Donenfeld [29]. A code review of the WireGuard Go code indicates that cookies are sent as soon as a queue, which is holding incoming handshakes, has reached a certain capacity. Future work could focus on coordinating the WireGuard cookie mechanism better with the used HSM for example by coordinating queue sizes and timeouts.

Molyneaux [70] argued in 2009 that performance testing is an informal discipline. Acquiring adequate literature on how to set up performance testing was hard to come by back then. From the experience of this thesis's author, this situation still needs improvement.

List of Figures

List of Tables

Glossary

`iperf` Iperf is a network performance measurement tool. It can create a data stream between two instances to measure throughput.

Go Go is a typed high-level programming language.

WireGuard WireGuard is a free and open-source virtual private network protocol.

Acronyms

AE	Authenticated Encryption.
AEAD	Authenticated Encryption with Additional Data.
ARPANET	Advanced Research Projects Agency Network.
CPU	Central Processing Unit.
DHKE	Diffie-Hellman Key Exchange.
ECDH	Elliptic Curve Diffie-Hellman Key Exchange.
EoIP	Ethernet over Internet Protocol.
FTP	File Transfer Protocol.
GAN	Global Area Network.
HKDF	HMAC-based Key Derivation Function.
HMAC	hash-based Message Authentication Code.
HSM	Hardware Security Module.
HTTP	Hypertext Transfer Protocol.
Hz	Hertz.
IoT	Internet of Things.
IP	Internet Protocol.
IPsec	Internet Protocol Security.
IS-IS	Intermediate System to Intermediate System Protocol.
ISO	International Standards Organization.

KDF	Key Derivation Function.
KMS	Key Management System.
L2TP	Layer 2 Tunneling Protocol.
LAN	Local Area Network.
MAC	Message Authentication Code.
MAN	Metropolitan Area Network.
MPLS	Multiprotocol Label Switching.
NIST	National Institute of Standards and Technology.
OSI	Open Systems Interconnection.
OSPF	Open Shortest Path First.
PAN	Personal Area Network.
PIN	Personal Identification Number.
PPTP	Point-to-Point Tunneling Protocol.
RSA	Rivest-Shamir-Adleman.
RTT	Round-Trip Time.
SMTP	Simple Mail Transfer Protocol.
SSH	Secure Shell.
SSL	Secure Socket Layer.
TLS	Transport Layer Security.
UDP	User Datagram Protocol.
VoIP	Voice over Internet Protocol.
VPN	Virtual Private Network.
WAN	Wide Area Network.
WANem	Wide Area Network Emulator.

Scientific Literature

[1] Adnan M. Abdulazeez et al. “Comparison of VPN Protocols at Network Layer Focusing on Wire Guard Protocol”. In: *International Journal of Interactive Mobile Technologies (iJIM)* 14.18 (Nov. 2020), pp. 157–177. DOI: 10.3991/ijim.v14i18.16507.

[2] Happy Akter et al. “Evaluating Performances of VPN Tunneling Protocols Based on Application Service Requirements”. In: *Proceedings of the Third International Conference on Trends in Computational and Cognitive Engineering*. Springer Nature Singapore, 2022, pp. 433–444. DOI: 10.1007/978-981-16-7597-3_36.

[3] Izzat Alsmadi et al. *Practical Information Security: A Competency-Based Education Course*. Springer International Publishing, 2018. ISBN: 978-3-319-72119-4. DOI: 10.1007/978-3-319-72119-4.

[4] Catalina Alvarez and Katerina Argyraki. “Using Gaming Footage as a Source of Internet Latency Information”. In: *Proceedings of the 2023 ACM on Internet Measurement Conference*. IMC ’23. Association for Computing Machinery, 2023, pp. 606–626. ISBN: 9798400703829. DOI: 10.1145/3618257.3624816.

[5] Jerzy Antoniuk and Małgorzata Plechawska-Wójcik. “Comparative analysis of VPN protocols”. In: *Journal of Computer Sciences Institute* 27 (July 2023), pp. 138–144. DOI: 10.35784/jcsi.3315.

[6] Jacob Appelbaum, Chloe Martindale, and Peter Wu. “Tiny WireGuard Tweak”. In: *Progress in Cryptology – AFRICACRYPT 2019*. Springer International Publishing, 2019, pp. 3–20. ISBN: 978-3-030-23696-0.

[7] Yazan Aref and Abdelkader Ouda. “HSM4SSL: Leveraging HSMs for Enhanced Intra-Domain Security”. In: *Future Internet* 16.5 (2024). ISSN: 1999-5903. DOI: 10.3390/fi16050148.

[8] Jean-Philippe Aumasson et al. “BLAKE2: Simpler, Smaller, Fast as MD5”. In: *Applied Cryptography and Network Security*. Springer Berlin Heidelberg, 2013, pp. 119–135. ISBN: 978-3-642-38980-1.

[9] Si T. Aung and Thandar Thein. “Comparative Analysis of Site-to-Site Layer 2 Virtual Private Networks”. In: *2020 IEEE Conference on Computer Applications (ICCA)*. 2020, pp. 1–5. DOI: 10.1109/ICCA49400.2020.9022848.

[10] Rajkumar Banoth and Rekha Regar. *Classical and Modern Cryptography for Beginners*. Springer, 2023. ISBN: 978-3-031-32959-3. DOI: 10.1007/978-3-031-32959-3_1.

[11] Graham Bath et al. *Foundation Level Specialist Syllabus: Performance Testing*. Tech. rep. Version 2018. International Software Testing Qualifications Board, 2018. URL: https://www.istqb.org/wp-content/uploads/2024/11/ISTQB-CT-PT_Syllabus_v1.0_2018.pdf (visited on 03/23/2025).

[12] Christian Baun. *Computer Networks / Computernetze*. Springer Fachmedien Wiesbaden, 2022. DOI: 10.1007/978-3-658-38893-5.

[13] Mihir Bellare, Ran Canetti, and Hugo Krawczyk. "Keying hash functions for message authentication". In: *Advances in Cryptology - CRYPTO '96*. Springer. Springer Berlin Heidelberg, 1996, pp. 1–15.

[14] Natalie Bennett. "Unveiling the Essence of Performance Testing: A Comprehensive Review". In: *Journal of Science & Technology* 3.4 (2022), pp. 41–51.

[15] Mario Bernhart and Roland Breiteneder. "Softwaretechnik - Mit Fallbeispielen aus realen Entwicklungsprojekten". In: 1st ed. München: Pearson Studium, 2009. Chap. 7, pp. 297–348. URL: https://www.inso-world.com (visited on 03/23/2025).

[16] Daniel J. Bernstein. *ChaCha, a variant of Salsa20*. Tech. rep. Document No. 2008/476. University of Illinois at Chicago, 2008. URL: https://cr.yp.to/chacha/chacha-20080128.pdf (visited on 03/23/2025).

[17] Daniel J. Bernstein. "Introduction to post-quantum cryptography". In: *Post-Quantum Cryptography*. Springer Berlin Heidelberg, 2009, pp. 1–14. DOI: 10.1007/978-3-540-88702-7_1.

[18] Matt Bishop. *Introduction to Computer Security*. Addison-Wesley Professional, 2004. ISBN: 9780321247442.

[19] Daniel P. Bovet and Marco Cesati. *Understanding the Linux kernel*. 3rd ed. O'Reilly Media, 2006. ISBN: 0596517386.

[20] Setiyo Budiyanto and Dadang Gunawan. "Comparative Analysis of VPN Protocols at Layer 2 Focusing on Voice Over Internet Protocol". In: *IEEE Access* 11 (2023), pp. 60853–60865. DOI: 10.1109/ACCESS.2023.3286032.

[21] James H. Carmouche. *IPsec virtual private network fundamentals*. Cisco Press, 2007. ISBN: 1-58705-207-5.

[22] Ramaswamy Chandramouli and Doron Pinhas. *Security Guidelines for Storage Infrastructure*. 2020. DOI: 10.6028/nist.sp.800-209.

[23] Choon H. Chua and Sok C. Ng. "Open-Source VPN Software: Performance Comparison for Remote Access". In: *Proceedings of the 5th International Conference on Information Science and Systems*. ICISS '22. Association for Computing Machinery, 2022, pp. 29–34. DOI: 10.1145/3561877.3561882.

[24] PCI Security Standards Council. *Payment Card Industry PTS HSM Modular Security Requirements*. Tech. rep. Version v4.0. 2021. URL: https://listings.pcisecuritystandards.org/documents/PCI_HSM_Security_Requirements_v4.pdf (visited on 03/23/2025).

[25] Common Criteria. *Common Criteria for Information Technology Security Evaluation*. Tech. rep. 2022. URL: https://www.commoncriteriaportal.org (visited on 03/23/2025).

[26] Michel Crouhy, Dan Galai, and Robert Mark. *The essentials of risk management*. McGraw-Hill Education Ltd, 2023. ISBN: 978-1-26-425887-1.

[27] Whitfield Diffie and Martin E. Hellman. "New directions in cryptography". In: *IEEE Transactions on Information Theory* 22.6 (1976), pp. 644–654. DOI: 10.1109/TIT.1976.1055638.

[29] Jason A. Donenfeld. *WireGuard: Next Generation Kernel Network Tunnel*. 2017. DOI: 10.14722/ndss.2017.23160.

[30] Benjamin Dowling and Kenneth G. Paterson. "A Cryptographic Analysis of the WireGuard Protocol". In: *Applied Cryptography and Network Security*. Springer International Publishing, 2018, pp. 3–21. DOI: 10.1007/978-3-319-93387-0_1.

[31] Claudia Eckert. *IT-Sicherheit: Konzepte - Verfahren - Protokolle*. Oldenbourg Wissenschaftsverlag, 2013. ISBN: 978-3-486-73587-1. DOI: 10.1524/9783486735871.bm.

[32] Ata Elahi and Alex Cushman. *Computer Networks*. Springer International Publishing, 2024. DOI: 10.1007/978-3-031-42018-4.

[33] Ludwig Fahrmeir et al. *Statistik: Der Weg zur Datenanalyse*. Springer Berlin Heidelberg, 2023. ISBN: 978-3-662-67526-7. DOI: 10.1007/978-3-662-67526-7_1.

[34] George Finney and John Kindervag. *Project Zero Trust: a story about a strategy for aligning security and the business*. Wiley, 2022. ISBN: 1119884845.

[35] Jason Garbis and Jerry W. Chapman. *Zero Trust Security: An Enterprise Guide*. Apress, 2021. ISBN: 9781484267011. DOI: 10.1007/978-1-4842-6702-8_1.

[36] Antonio F. Gentile et al. "A VPN Performances Analysis of Constrained Hardware Open Source Infrastructure Deploy in IoT Environment". In: *Future Internet* 14.9 (2022). DOI: 10.3390/fi14090264.

[37] Kinan Ghanem et al. "Security vs Bandwidth: Performance Analysis Between IPsec and OpenVPN in Smart Grid". In: *2022 International Symposium on Networks, Computers and Communications (ISNCC)*. 2022, pp. 1–5. DOI: 10.1109/ISNCC55209.2022.9851717.

[38] Tom Goethals et al. “Scalability evaluation of VPN technologies for secure container networking”. In: *2019 15th International Conference on Network and Service Management (CNSM)*. 2019, pp. 1–7. DOI: 10.23919/CNSM46954.2019.9012673.

[39] Brendan Gregg. *Systems Performance*. 2nd ed. Pearson Education, 2021. ISBN: 978-0-13-682015-4.

[40] *Guide for conducting risk assessments*. 2012. DOI: 10.6028/nist.sp.800-30r1.

[41] Ashwin Gumaste. “Metropolitan Networks”. In: *Springer Handbook of Optical Networks*. Springer International Publishing, 2020, pp. 609–630. ISBN: 978-3-030-16250-4. DOI: 10.1007/978-3-030-16250-4_18.

[42] Juhyeng Han et al. “Scalable and Secure Virtualization of HSM With ScaleTrust”. In: *IEEE/ACM Transactions on Networking* 31.4 (2023), pp. 1595–1610. DOI: 10.1109/TNET.2022.3220427.

[43] Juhyeng Han et al. “Toward scaling hardware security module for emerging cloud services”. In: *Proceedings of the 4th Workshop on System Software for Trusted Execution*. SysTEX ’19. Association for Computing Machinery, 2019. ISBN: 978-1-450-36888-9. DOI: 10.1145/3342559.3365335.

[44] Gilbert Held. *Virtual private networking: a construction, operation and utilization guide*. John Wiley, 2004. ISBN: 0-470-85432-4.

[46] “IEEE Approved Draft Standard for Ethernet Amendment: Media Access Control Parameters for 800 Gb/s and Physical Layers and Management Parameters for 400 Gb/s and 800 Gb/s Operation”. In: *IEEE P802.3df/D3.2, November 2024* (2024), pp. 1–301.

[47] ISO Central Secretary. *Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - Product quality model*. Standard ISO/IEC TR 25010:2023(E). Geneva, CH: International Organization for Standardization, 2023.

[48] Johan Ivarsson and Andreas Nilsson. *A review of hardware security modules: Fall 2010*. Tech. rep. 2010.

[49] Zhen M. Jiang and Ahmed E. Hassan. “A Survey on Load Testing of Large-Scale Software Systems”. In: *IEEE Transactions on Software Engineering* 41.11 (2015), pp. 1091–1118. DOI: 10.1109/TSE.2015.2445340.

[50] Karuna K. Jyothi and Indira B. Reddy. “Study on virtual private network (VPN), VPN’s protocols and security”. In: *International Journal of Scientific Research in Computer Science, Engineering and Information Technology* 3.5 (2018), pp. 919–932.

[51] Ashvin Kamaraju, Asad Ali, and Rohini Deepak. “Best Practices for Cloud Data Protection and Key Management”. In: *Proceedings of the Future Technologies Conference (FTC) 2021, Volume 3*. Springer International Publishing, 2022, pp. 117–131. ISBN: 978-3-030-89912-7.

[52] Martin Kappes. *Netzwerk- und Datensicherheit: Eine praktische Einführung*. Teubner, 2007. ISBN: 978-3-8351-9202-7. DOI: `10.1007/978-3-8351-9202-7`.

[53] Joseph M. Kizza. *Guide to Computer Network Security*. 6th ed. Springer Nature Switzerland, 2024. ISBN: 978-3-031-47548-1. DOI: `10.1007/978-3-031-47549-8`.

[54] Frank H. Knight. *Risk, uncertainty and profit*. Houghton Mifflin, 1921.

[55] Hugo Krawczyk. “Cryptographic Extraction and Key Derivation: The HKDF Scheme”. In: *Advances in Cryptology – CRYPTO 2010*. Springer Berlin Heidelberg, 2010, pp. 631–648. ISBN: 978-3-642-14623-7.

[56] Panagiotis Leloudas. *Introduction to Software Testing*. Apress, 2023. DOI: `10.1007/978-1-4842-9514-4`.

[57] David M. Levine and David F. Stephan. *Even You Can Learn Statistics*. Addison-Wesley Professional, 2022. ISBN: 978-0-13-701059-2.

[58] David J. Lilja. *Measuring Computer Performance: A Practitioner’s Guide*. Cambridge University Press, 2000. ISBN: 978-0-521-64105-0. DOI: `10.1017/CBO9780511612398`.

[59] Susan Lincke. *Information Security Planning: A Practical Approach*. Springer International Publishing, 2024. ISBN: 978-3-031-43117-3. DOI: `10.1007/978-3-031-43118-0`.

[60] Benjamin Lipp, Bruno Blanchet, and Karthikeyan Bhargavan. “A Mechanised Cryptographic Proof of the WireGuard Virtual Private Network Protocol”. In: *2019 IEEE European Symposium on Security and Privacy*. 2019, pp. 231–246. DOI: `10.1109/EuroSP.2019.00026`.

[61] Steven Mackey et al. “A Performance Comparison of WireGuard and OpenVPN”. In: *Proceedings of the Tenth ACM Conference on Data and Application Security and Privacy*. CODASPY ’20. Association for Computing Machinery, 2020, pp. 162–164. DOI: `10.1145/3374664.3379532`.

[62] Alexander Master and Christina Garman. “A WireGuard Exploration”. In: *CERIAS Technical Reports* (2023). DOI: `10.5703/1288284317610`.

[63] Sai Matam and Jagdeep Jain. *Pro Apache JMeter: Web Application Performance Testing*. Apress, 2017. DOI: `10.1007/978-1-4842-2961-3`.

[64] Justin Matejka and George Fitzmaurice. “Same Stats, Different Graphs: Generating Datasets with Varied Appearance and Identical Statistics through Simulated Annealing”. In: *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*. CHI ’17. Association for Computing Machinery, 2017, pp. 1290–1294. ISBN: 9781450346559. DOI: `10.1145/3025453.3025912`.

[65] Stathis Mavrovouniotis and Mick Ganley. “Hardware Security Modules”. In: *Secure Smart Embedded Devices, Platforms and Applications.* Springer New York, 2014, pp. 383–405. DOI: 10.1007/978-1-4614-7915-4_17.

[67] Alfred J. Menezes, Paul C. Van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography.* CRC press, 2018. ISBN: 978-0-84-938523-0. DOI: 10.1201/9780429466335.

[68] *Minimum security requirements for federal information and information systems.* 2006. DOI: 10.6028/nist.fips.200.

[69] Jibitesh Mishra and Ashok Mohanty. *Software engineering.* 1st ed. Dorling Kindersley, 2011. URL: https://learning.oreilly.com/library/view/software-engineering/9788131758694/?ar= (visited on 03/23/2025).

[70] Ian Molyneaux. *The art of application performance testing.* 1st ed. O’Reilly Media, Inc., 2009. ISBN: 978-0-596-52066-3.

[71] Claude M. Nawej and Shengzhi Du. “Virtual Private Network’s Impact on Network Performance”. In: *2018 International Conference on Intelligent and Innovative Computing Applications (ICONIC).* 2018, pp. 1–6. DOI: 10.1109/ICONIC.2018.8601281.

[72] Gerard O’Regan. *Concise Guide to Software Engineering: From Fundamentals to Application Methods.* Springer International Publishing, 2022. ISBN: 978-3-031-07816-3. DOI: 10.1007/978-3-031-07816-3.

[73] Mohammad S. Obaidat and Noureddine Boudriga. *Fundamentals of performance evaluation of computer and telecommunications systems.* John Wiley and Sons, 2010. ISBN: 978-0-471-26983-0.

[74] David Parkinson. *Risk: Analysis, perception and management. Report of a Royal Society Study Group.* 1993.

[75] Larry L. Peterson and Bruce S. Davie. *Computer Networks: A Systems Approach.* 6th ed. Elsevier Science and Technology, 2019. URL: https://github.com/SystemsApproach/book (visited on 03/23/2025).

[76] *Prevention through Design Guidelines for Addressing Occupational Hazards and Risks in Design and Redesign Processes.* American National Standards Institute / American Society of Safety Professionals, 2011.

[77] Umesh H. Rao and Umesha Nayak. *The InfoSec Handbook: An Introduction to Information Security.* Apress, 2014. ISBN: 978-1-4302-6383-8. DOI: 10.1007/978-1-4302-6383-8.

[78] Marvin Rausand and Stein Haugen. *Risk Assessment: Theory, Methods, and Applications.* Wiley, 2020. ISBN: 978-1-11-937735-1. DOI: 10.1002/9781119377351.ch2.

[80] Matthew N. O. Sadiku and Cajetan M. Akujuobi. *Fundamentals of Computer Networks.* Springer International Publishing, 2022. ISBN: 978-3-031-09417-0. DOI: 10.1007/978-3-031-09417-0.

[81] Matthew N. O. Sadiku and Sarhan M. Musa. *Performance Analysis of Computer Networks*. Springer International Publishing, 2013. ISBN: 978-3-319-01646-7. DOI: 10.1007/978-3-319-01646-7.

[82] Bruce Schneier. *Secrets & Lies: Digital Security in a Networked World*. John Wiley & Sons, Inc., 2000. ISBN: 978-0-471-25311-2.

[83] Jörg Schwenk. *Guide to Internet Cryptography: Security Protocols and Real-World Attack Implications*. Springer International Publishing, 2022. ISBN: 978-3-031-19439-9. DOI: 10.1007/978-3-031-19439-9.

[84] Naresh K. Sehgal, Pramod C. P. Bhatt, and John M. Acken. Springer International Publishing, 2022. ISBN: 978-3-031-07242-0. DOI: 10.1007/978-3-031-07242-0.

[85] Robin Sharp. *Introduction to Cybersecurity: A Multidisciplinary Challenge*. Springer Nature Switzerland, 2024. ISBN: 978-3-031-41463-3. DOI: 10.1007/978-3-031-41463-3.

[86] Maria Sommerhalder. "Hardware Security Module". In: *Trends in Data Protection and Encryption Technologies*. Springer Nature Switzerland, 2023, pp. 83–87. DOI: 10.1007/978-3-031-33386-6_16.

[87] National Institute of Standards and Technology. *FIPS PUB 140-2: Security Requirements for Cryptographic Modules*. Tech. rep. FIPS PUB 140-2. U.S. Department of Commerce, 2001.

[88] Lucjan Stapp, Adam Roman, and Michaël Pilaeten. *ISTQB® Certified Tester Foundation Level: A Self-Study Guide Syllabus v4.0*. Springer Nature Switzerland, 2024. ISBN: 978-3-031-42767-1. DOI: 10.1007/978-3-031-42767-1.

[89] Chandramouli Subramanian et al. *Software engineering*. 1st ed. Pearson India Education Services, 2016. URL: https://learning.oreilly.com/library/view/software-engineering/9789332558298/?ar= (visited on 03/23/2025).

[90] Chellammal Surianarayanan and Pethuru R. Chelliah. Springer International Publishing, 2023. ISBN: 978-3-031-32044-6. DOI: 10.1007/978-3-031-32044-6.

[91] Andrew S. Tanenbaum and David J. Wetherall. *Computer networks*. 5th ed. Pearson, 2011. ISBN: 978-0-13-212695-3.

[92] Yu-Chu Tian and Jing Gao. *Network Analysis and Architecture*. Springer Nature Singapore, 2024. ISBN: 978-981-99-5648-7. DOI: 10.1007/978-981-99-5648-7.

[93] Minh-Tuan Truong and Quang-Vinh Dang. "Digital Signatures Using Hardware Security Modules for Electronic Bills in Vietnam: Open Problems and Research Directions". In: *Future Data and Security Engineering. Big Data, Security and Privacy, Smart City and Industry 4.0 Applications*. Springer Singapore, 2020, pp. 469–475. ISBN: 978-981-33-4370-2.

[94] *U.S. Code Title 44 - PUBLIC PRINTING AND DOCUMENTS*. 2013.

[96] Pascal Urien. "A New Approach for Crypto Off-loading Based on Personal HSM". In: *2023 7th Cyber Security in Networking Conference*. 2023, pp. 23–26. DOI: 10.1109/CSNet59123.2023.10339762.

[97] Paul C. van Oorschot. *Computer Security and the Internet*. Springer International Publishing, 2021. DOI: 10.1007/978-3-030-83411-1.

[98] Elaine J. Weyuker and Filippos I. Vokolos. "Experience with performance testing of software systems: issues, an approach, and case study". In: *IEEE Transactions on Software Engineering* 26.12 (2000), pp. 1147–1156. DOI: 10.1109/32.888628.

[99] Claus O. Wilke. *Fundamentals of data visualization: a primer on making informative and compelling figures*. O'Reilly Media, 2019. ISBN: 978-1-492-03108-6.

[100] David Wong. *Real-world cryptography*. Manning Publications Co., 2021. ISBN: 978-1-61729-671-0.

[101] Keith Yorkston. *Performance Testing*. Apress, 2021. DOI: 10.1007/978-1-4842-7255-8.

Online References

[28] Jason A. Donenfeld. *wg-quick*. Version v1.0.20210914. 2021. URL: `https://github.com/WireGuard/wireguard-tools` (visited on 03/23/2025).

[45] John Hunter. *matplotlib*. Version 3.8.3. 2024. URL: `https://github.com/matplotlib/matplotlib` (visited on 03/23/2025).

[66] Robert McMahon. *iperf*. Version 2.1.5. 2021. URL: `https://sourceforge.net/projects/iperf2/` (visited on 03/23/2025).

[79] *RIPE Altas*. URL: `https://atlas.ripe.net/` (visited on 03/23/2025).

[95] *uptime man page*. URL: `https://man7.org/linux/man-pages/man1/uptime.1.html` (visited on 03/23/2025).

Overview of Generative AI Tools Used

1. **Google Translate**
   - Usage: Was used as a starting point to translate more complex ideas from the native language into the thesis language.
   - Place of use: Whole thesis.
2. **Grammarly Free Version**
   - Usage: Was used for grammar and spell-checking.
   - Place of use: Whole thesis.