12-13-2023 10:30 AM

SSL Everywhere: Leveraging HSMs for Enhanced Intra-Domain Security

Yazan Aref, *Western University*

Supervisor: Ouda, Abdelkader H., *The University of Western Ontario*

A thesis submitted in partial fulfillment of the requirements for the Master of Engineering Science degree in Electrical and Computer Engineering

Abstract

In a world where digitalization is rapidly advancing, the security and privacy of intra-domain communication within organizations are of critical concern. The imperative to secure communication channels among physical systems has led to the deployment of various security approaches aimed at fortifying networking protocols. However, these approaches have typically been designed to secure protocols individually, lacking a holistic perspective on the broader challenge of intra-domain communication security. This omission raises fundamental concerns about the safety and integrity of intra-domain environments, where all communication occurs within a single domain. As a result, this thesis introduces SSL Everywhere, a comprehensive solution designed to address the evolving challenges of secure data transmission in intra-domain environments. By leveraging Hardware Security Modules (HSMs), SSL Everywhere aims to utilize the Secure Socket Layer (SSL) protocol within intra-domain environments to ensure data confidentiality, authentication, and integrity.

In addition, solutions proposed by academic researchers and industry have not addressed the issue in a holistic and integrative manner, as they only apply to specific types of environments or servers, and do not utilize all cryptographic operations for robust security. Thus, SSL Everywhere bridges this gap by offering a unified and comprehensive solution that includes certificate management, key management practices, and various security services.

By acknowledging the importance of secure communication principles and their application within the unique context of intra-domain communication, this research contributes to the ongoing discourse on network security and provides a promising pathway to secure the future of intra-domain environments.

Keywords: Intra-Domain Communication, Computer Networks, Network Security, Hardware Security Module, Certificate Management, Secure Sockets Layer, Certificate Authority.

Summary For Lay Audience

In today's digital age, ensuring the security of our online communication has become more critical than ever. One area where this security is paramount is within organizations, where multiple servers are placed to handle various requests. The optimal solution to ensure secure communication between those servers is to deploy Secure Socket Layer (SSL). SSL is a fundamental security protocol that helps transform readable data into an unreadable format during transmission. Moreover, Deploying SSL in a communication channel verifies the identities of the two parties communicating with each other.

While SSL is a powerful protocol, implementing it within the communication channels in organizational domains can be quite challenging. SSL demands extensive cryptographic practices to ensure that cryptographic keys and digital certificates are correctly configured and up to date. This complexity often leaves organizations grappling with the complexities of SSL deployment.

Thus, multiple security mechanisms for the currently employed networking protocols have been proposed as potential alternatives to SSL. Plus, multiple solutions have been presented to overcome the issue of securing organizational communication. However, those solutions always have a gap in addressing the comprehensive security needs of intra-domain environments. These environments, where all communication occurs within a single domain, such as an organization, demand a unified approach to ensure communication security. The existing security mechanisms typically focus on securing individual aspects and lack the comprehensive protection required for sensitive data transmission.

As a result, this thesis introduces SSL Everywhere, a standardized solution designed to address the evolving challenges of secure data transmission in intra-domain environments. SSL Everywhere leverages Hardware Security Modules (HSMs), which are hardware devices that act as a root of trust to perform various cryptographic operations, to fully utilize SSL protocol within intra-domain environments.

Acknowledgements

First and Foremost, Alhamdulillah. I praise Allah (SWT) the Almighty, the Most Gracious, and the Most Merciful for granting me strength, guidance, and wisdom throughout this journey and in completing this thesis.

Secondly, I extend my heartfelt appreciation to my family for their unwavering support and encouragement. Their love and belief in me have been my pillars of strength, and I am forever grateful for their sacrifices and understanding during this endeavor.

I am indebted and I would like to express my deepest gratitude to my thesis advisor, Dr. Abdelkader Ouda, for his invaluable guidance, unwavering support, and expertise throughout the entire research process. His mentorship and insightful feedback have been instrumental in shaping this work.

Finally, I would like to thank my friends, the dedicated members of the research team, my esteemed course supervisor, and the administrative staff for their collective cooperation throughout this journey.

Contents

List of Tables

List of Figures

List of Abbreviations

AS	Autonomous System
SSL	Secure Socket Layer
DES	Data Encryption Standard
RSA	Rivest-Shamir-Adleman
CA	Certificate Authority
CSR	Certificate Signing Request
HSM	Hardware Security Modules
FIPS	Federal Information Processing Standard
NREL	National Renewable Energy Laboratory
KMS	Key Management Service
SaaS	Software as a Service
ADCP	Adaptive Driver Communication Protocol
HAL	Hardware Abstraction Layer
API	Application Programming Interface
MAC	Media Access Control
IP	Internet Protocol
OpenSSL	Open Secure Socket Layer
TOP	Table Of Processes
RPS	Requests Per Second

Chapter 1

Introduction

In an era defined by rapid technological advancement and increasing digitalization, security and privacy have emerged as paramount concerns for individuals, businesses, and organizations worldwide. The seamless exchange of sensitive information across networks and the Internet has necessitated the development of robust mechanisms to protect data integrity, confidentiality, and authenticity. Consequently, a great deal of work has gone into making internet communication channels secure. However, as the digital landscape evolves and networks expand to encompass a multitude of devices, a new dimension of security challenges arises. This is particularly evident in intra-domain environments, where organizations face a growing need to secure communication within their own networks. In an intra-domain environment, all communications are confined within the parameters of a single *Autonomous System* (AS). This term refers to a collection of interconnected physical network components, such as servers, data centers, and other network devices that are administered and overseen by one organizational domain. Such an environment is characterized by the internal exchange of information and services like email and web browsing, without extending beyond the organization's network boundaries to other domains [1]. Figure 1.1 shows an example of intra-domain communication within autonomous systems as well as inter-domain communication between them. Augmenting intra-domain networks' capacities to include many devices with network capabilities raises the importance of securing communication in those environments, as in 2020, 68% of ma-

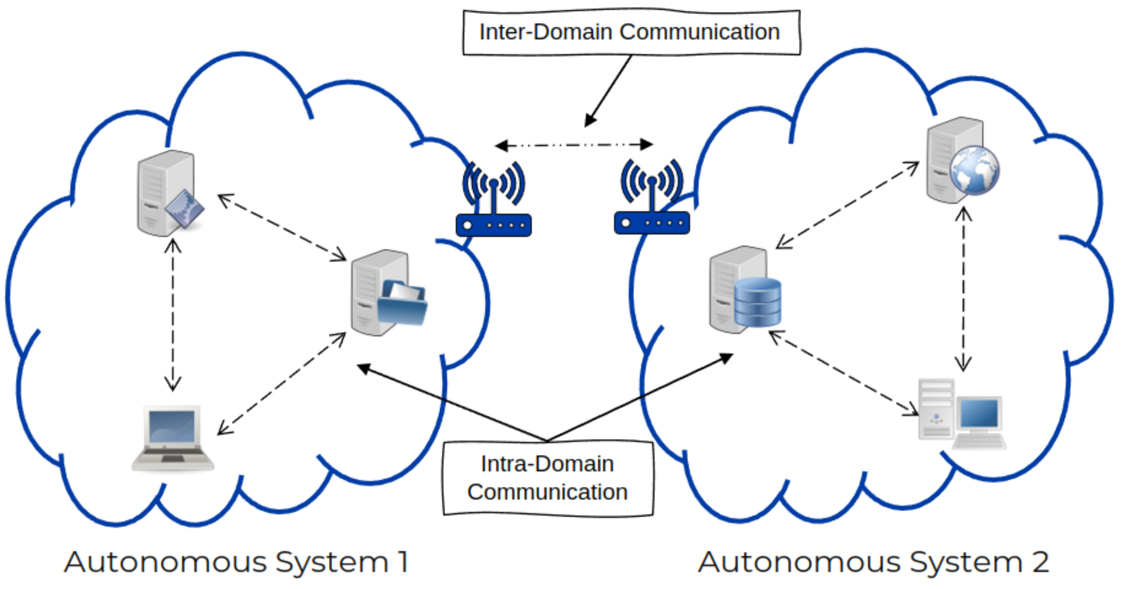


Figure 1.1: Intra-Domain Communication Within Autonomous Systems

licious activities and security-related incidents within organizations are attributed to insider employees, often with compromised devices [2]. This underscores the urgency of addressing security concerns beyond the public Internet.

Servers within intra-domain environments are compelled to communicate with clients and peers to exchange resources; hence, each server requires a mechanism to secure this communication. Thus, various security mechanisms are employed to supposedly secure those communication channels. Unfortunately, however, those mechanisms lack sufficient protection to encrypt data and authenticate endpoints, opening the floor to attacks from compromised and misconfigured machines.

This thesis is focused on addressing and pointing out the current insecurities intra-domain communication is facing and proposing a novel architecture to fortify communication channels within organizational networks.

1.1 Motivation

Primarily, organizations that employ multiple servers to handle various requests utilize *load balancers* or *reverse proxies* to intercept traffic emanating from multiple hosts or endpoints, separate the received requests and distribute them across multiple back-end servers as demonstrated in Figure 1.2. The load balancing process is vital for all cloud environments as it is proven to enhance reliability and availability by monitoring servers' statuses and directing requests to available servers only [3], and relieving back-end servers from the tiring process of decrypting traffic by employing *Secure Socket Layer (SSL) Termination*, which also aids intrusion detection systems to detect anomaly packets before forwarding them to the servers [4]. However, this process poses security risks as it creates insecure channels between the reverse proxy and the back-end servers, leaving communication vulnerable to various types of attacks.

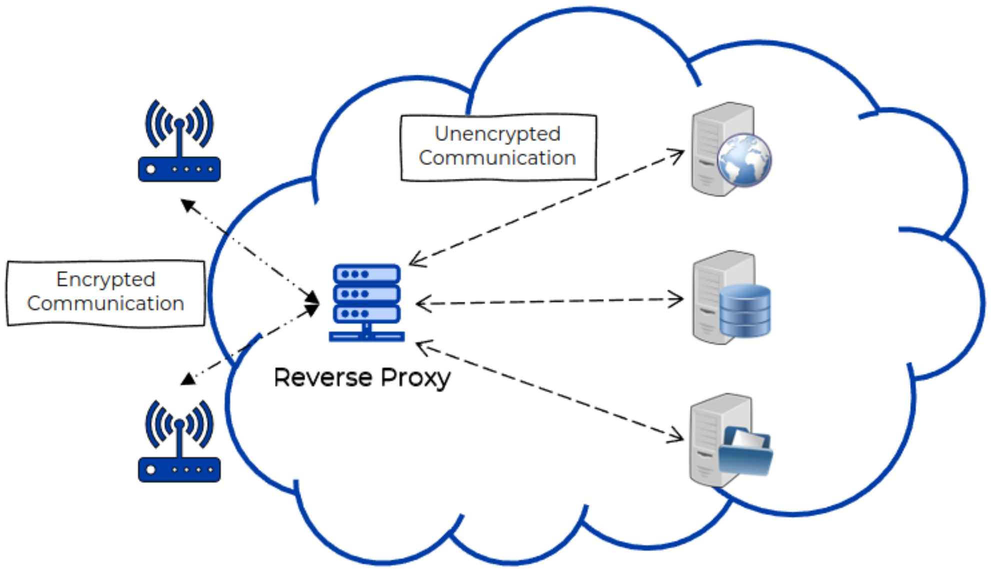


Figure 1.2: Reverse Proxy Example

Furthermore, since 2016, the Certificate Authority CA/Browser Forum, responsible for setting standards in the issuance and management of SSL certificates, has ceased issuing SSL certificates for servers with private IP addresses. This implies that servers within an intra-domain environment are unable to meet SSL communication requirements for encryption and endpoint authentication [5].

Another concern arises from the intricate task of managing certificates for servers with dynamic IP addresses, which presents a complex and challenging task. In scenarios where servers are assigned new IP addresses dynamically, it requires the issuance of a new certificate each time the IP address changes. This dynamic nature of IP addresses, common in many modern network configurations, can quickly lead to certificate management complexities. The traditional approach of generating and deploying certificates for each IP address change not only introduces administrative overhead but also raises questions about certificate validity, revocation, and the seamless continuity of secure communication [6].

Along with the aforementioned issues, the current development of secure intra-domain communication is being underrepresented by both academia and implemented systems, with reasons still being obscure. Against this background, this situation is continuously giving rise to countless security breaches and attacks within organizations, making most intra-domain attacks rise from the fact that intra-domain communication; particularly between servers remains, by and large, unencrypted and unauthenticated [7][8]. However, Certificate usage for SSL establishment between the load balancer and the servers and also between all endpoints was suggested by *Boisrond* in [9] as the optimum solution to address the insecure communication issue.

Having said the above, we believe that there is a significant opportunity to develop an innovative architecture that addresses these challenges. Such an architecture would streamline the process of securing communication within intra-domain environments. Our architecture would not only enhance operational efficiency and security but also reduce the complexity currently associated with securing intra-domain communication. By tackling these issues head-on, the proposed solution aims to provide a more seamless, secure, and scalable approach to secure internal networks.

Morevoer, delving previously into the research on autonomous vehicles cyber-attacks highlighted the critical importance of securing digital communication channels [10]. This prior work served as a strong motivator to explore security solutions in the realm of network communication. Recognizing the critical need to protect communication within intra-domain environments, efforts were aimed at extending the research to address evolving challenges and potential vulnerabilities within such environments.

1.2 Objectives

The main objective of this research is to build a standardized architecture that addresses the contemporary security challenges within intra-domain environments. This architecture is designed to reduce the complexities associated with securing data exchange within organizational networks, streamline certificate management for servers, and provide a unified solution that enhances communication security while ensuring seamless continuity. The specific sub-objectives aligned with this main objective are as follows:

1. Conduct a comprehensive study and analysis of existing security solutions and practices within intra-domain environments, assessing their strengths, weaknesses, and applicability.

2. Design and implement a unified security framework for intra-domain environments that offers security services, including encryption, authentication, and data integrity measures, ensuring comprehensive protection of data in transit.

3. Assess the performance of the proposed architecture in accommodating scaling network traffic and communication demands, with a focus on minimizing latency and maximizing throughput.

1.3 Research Methodology

This section outlines the methodologies in this research based on the pre-defined objectives.

The first objective contains a comprehensive study of the strengths and weaknesses of current intra-domain security mechanisms. It includes the following:

- A thorough review of the current security solutions and protocols in intra-domain environments, assessing their implementation and effectiveness.

- An analysis of the strengths, vulnerabilities, and limitations of these security measures provides a critical perspective on their practical applicability.

- A comparison of recently developed security technologies and approaches.
- Pointing out specific areas where current security solutions fall short, setting the stage for the development of our architecture.

Moving on to the second objective, it is concerned with proposing and developing a standardized architecture to overcome the indicated weaknesses. This includes:

- Propose developing an integrated structure that provides various security services for intra-domain environments, encompassing diverse environmental characteristics. This includes securing communication at the application level, ensuring robust protection between servers within the same network, and extending security measures to cover interactions between different networks.
- Design the architecture of the framework, detailing the integration of the various components within the architecture.
- Define the security services provided by the architecture.
- Implement and build the proposed architecture in a bottom-up approach.

The last objective is to evaluate the performance of the developed architecture by completing the following tasks:

- Choose the appropriate metrics to be evaluated.
- Choose a conventional approach to compare with.
- Choose the appropriate software tools. to perform the assessments.
- Analyze the results and propose solutions to overcome weaknesses.

This structured approach provides a clear roadmap for how each objective is achieved. Moreover, Chapter 3, Chapter 4, and Chapter 5 explain the methodology in more details.

1.4 Outline

This thesis is structured as follows, Chapter 2 provides a discussion on commonly used networking protocols within organizational networks along with their security measures. These protocols are examined in detail to clarify their characteristics and underscore their inherent vulnerabilities. Plus, an exploration of prior research endeavors is undertaken, encompassing existing solutions aimed at addressing these issues. Subsequently, Chapter 3 delves into a comprehensive presentation and detailed discussion of our architecture, including the roles and functionalities of its integral components. Then, we shed light on the design process of our architecture and provide insights into its development and underlying principles in Chapter 4. In Chapter 5, a comprehensive performance assessment and analysis of the architecture are presented, including a detailed discussion of the achieved results. This chapter also explores methods for enhancing the architecture and addressing its inherent weaknesses. In conclusion, Chapter 6 provides the final insights and conclusions drawn from the research, offering a comprehensive summary of the key findings, contributions, and implications of the study.

Chapter 2

Background & Literature Review

The development of security mechanisms to secure individual networking protocols for communication within one AS or multiple ASs has been a focal point of research efforts. However, hardly any literature considered the problem in a comprehensive and holistic manner [8]. In this chapter, we provide a background on the essential components on the key elements and concepts that form the basis of our research. Plus, we offer a comprehensive overview of the networking protocols currently deployed, along with an examination of their accompanying security mechanisms. We also delve into an analysis of their vulnerabilities. Furthermore, we explore solutions put forth by both academic and industrial realms, which have made contributions to enhancing security in intra-domain environments. This chapter lays the foundation for understanding the context and significance of our architecture in addressing these challenges.

2.1 Background

In what follows, we provide essential context for understanding the foundation upon which our study is built. It includes secure communication principles, which form the basis for ensuring authenticity, data confidentiality and integrity during data transmission. Key elements of this foundation include the use of SSL, a cryptographic protocol that plays a pivotal role in securing data exchange over computer networks, particularly the Internet. Furthermore, the section delves into the significance of X.509 certificates, which are fundamental to authenticating the identity of certificate holders and establishing trust in secure communication. Additionally, we explore Hardware Security Modules, which are dedicated hardware devices that play a crucial role in secure key management and cryptographic operations that are used in our architecture.

2.1.1 Secure Communication Principles

Secure communication principles are the foundation of protecting data during transmission across networks. These principles include three fundamental elements: privacy, authentication, and integrity [11].

Privacy

Firstly, in the context of computer networks, particularly in an era characterized by an increasing dependence on computer networks and the Internet, data privacy emerges as a paramount concern for safeguarding user rights and preserving information security. At its core, data privacy entails the protection of sensitive information from unauthorized access or disclosure. Data privacy rights are fundamental in ensuring that individuals' personal information is handled responsibly and that they have the autonomy to decide how their data is used and shared. Encryption, a powerful method in the realm of privacy, helps transform data into an unreadable format (ciphertext) to ensure that it remains confidential and secure, further enhancing the safeguarding of privacy. Various methods are utilized for this purpose, including classic techniques like *Data Encryption Standard* (DES) and modern approaches such as *Rivest-Shamir-Adleman* (RSA) [12].

Authentication

Similar to encryption, authentication also stands as a crucial component in communication security, serving the essential purpose of verifying the identities of connecting entities. It involves the validation of provided credentials, such as usernames and passwords, digital certificates, or other authentication tokens, to ascertain the legitimacy of the connecting party. Effective server authentication not only safeguards sensitive data but also helps prevent unauthorized access attempts [13].

Integrity

Finally, integrity refers to the quality or state of being whole, unaltered, and uncorrupted. It encompasses the assurance that data, files, or information have not been tampered with, altered, or modified by unauthorized parties during storage, transmission, or processing. Data integrity ensures that information remains accurate, complete, and reliable, maintaining its original state and consistency [14].

2.1.2 Secure Socket Layer (SSL)

These communication principles are the fundamental building blocks for SSL. SSL, which stands for *Secure Sockets Layer*, is a cryptographic protocol used to secure data transmission over computer networks, particularly the Internet. SSL was developed to provide a secure and encrypted communication channel between a client and a server. It ensures that data exchanged between the client and server remains confidential and protected from eavesdropping or tampering by malicious actors. Successful SSL implementation achieves the three essential security attributes; encryption, authentication, and integrity for all exchanged communication.

The usage of SSL depends on digital certificates (e.g. X.509 certificates) to authenticate endpoints and exchange keys to encrypt/decrypt future communication. To perform mutual authentication between two endpoints, each endpoint involved in SSL communication should possess a digital certificate that contains information about its identity and a public key. These digital certificates are issued and validated by a *Certificate Authority* (CA), which is a trusted third party that is responsible for generating digital certificates for endpoints.

In addition to authentication, digital certificates facilitate the secure exchange of encryption keys between the client and the server. During the SSL handshake, a process that initiates a secure connection, the client and server exchange cryptographic parameters, negotiate encryption algorithms, and validate each other's certificates. Subsequently, they agree on a shared secret key, which is used for encrypting and decrypting future communication. This key exchange process ensures that even if a malicious entity intercepts the data, it cannot decipher the content without the appropriate key [15],[16].

2.1.3 X.509 Certificates

While SSL serves as a critical protocol for securing data transmission over networks, its effectiveness relies on the utilization of X.509 certificates. X.509 certificates are a standardized format of digital certificates that form the basis of the SSL protocol. It plays a crucial role in verifying the identity of an endpoint and establishing encrypted and authenticated communication channels over computer networks. Moreover, it also includes the certificate chain, which represents a hierarchical sequence of certificates that establishes a trust path from the certificate being presented to a trusted root certificate. This chain of certificates, also known as the certification path, allows clients to verify the authenticity of the presented certificate by following the digital signatures and relationships between certificates in the chain.

The root certificate at the top of this chain is inherently trusted, forming the basis of trust for the entire certificate infrastructure. This mechanism ensures that the certificate holder can be reliably authenticated, and the communication can proceed with confidence in its security and integrity [16],[17].

The generation of an X.509 certificate involves a *Certificate Signing Request* (CSR). CSRs play an important role, as they are utilized to gather essential information from a certificate applicant (client), such as their public key, distinguished name, and optional organizational information. Once collected, this information is encapsulated within the CSR, which is then submitted to a CA for signing. The CA's digital signature on the CSR results in the creation of an X.509 certificate [17].

2.1.4 Hardware Security Module

The security of cryptographic keys used in SSL and the protection of sensitive data are bolstered by *Hardware Security Modules* (HSMs), which are tamper-resistant hardware devices designed to provide secure key storage, cryptographic operations, and protection of sensitive data. HSMs enhance the security of cryptographic operations by securely generating, storing, and managing cryptographic keys. They perform cryptographic operations such as encryption, decryption, digital signatures, and secure hash functions within a secure boundary, protecting keys from external attacks. HSMs also feature built-in random number generators, resistance to physical tampering, compliance with cryptographic standards, and secure key backup and recovery mechanisms. These devices are isolated from host systems and are widely used in security-critical environments to safeguard sensitive data and ensure compliance with security regulations.

HSMs comply with the *Federal Information Processing Standard* (FIPS) 140-2 which is a standard for developing HSMs. It mandates HSMs to prevent unauthorized access to sensitive data, provide up-to-date cryptographic algorithms, and undergo a series of rigorous testing and evaluation. The FIPS 140-2 has four security levels, with the higher levels offering more protective features, where level 1 provides the lowest degree of security and lists basic security requirements, whereas level 4 provides a complete envelope of protection intended to detect and respond to all unauthorized physical attacks [18].
For an HSM to become a FIPS 140-2 compliant, it must employ the FIPS 140-2 algorithms, key management techniques, and authentication procedures. These three key areas encompass a wide range of design and implementation security requirements, including, but not limited to, those associated with:

- Cryptographic module ports and interfaces: It states that a cryptographic module must have four logical interfaces: the data input interface, the data output interface, the control input interface, and the status output interface. These account for incoming data, outgoing data, module operation controls, and output signals and status indicators, respectively.

- Roles, services, and authentication: It states that a cryptographic module must support a user role and a crypto officer role, a service output that shows the status of a module, and either role-based authentication or identity-based authentication mechanisms, depending on the security level.

- Physical security and environmental failure protection and testing: It requires a tamper detection or response envelope with tamper response and zeroization circuitry, which is circuitry that facilitates the automatic erasure of sensitive information, such as plaintext data and cryptographic keys, if the module is tampered with or stolen.[19]

2.2 Literature Review

Having established the foundational concepts of our research, this section explores the realm of established secure networking protocols, discussing their inner workings, strengths, and vulnerabilities. This analysis provides insights into the current state of secure communication technologies. Furthermore, we examine both academic and industrial contributions that have aimed to address the challenges posed by intra-domain communication weaknesses.

2.2.1 Existing Secure Networking Protocols

This section dives into discussing the details of some of the most widely adopted networking protocols, each playing a pivotal role in secure data transmission within intra-domain environments.

HTTP

Hypertext Transfer Protocol (HTTP) is a client-server and *Peer-to-Peer* (P2P) protocol that is one of the most utilized protocols today for data communication for the World Wide Web [20]. Employing HTTP for communication on the Internet allowed the exchange of information in the clear. Thus, the increase of sensitive application data has led to a debate on how clients and servers can communicate securely which obliged the improvement to a more secure version. HTTPS is the secure version of HTTP that engages SSL protocol to exchange HTTP messages between the client and the server in an encrypted and authenticated channel to protect from eavesdropping, spoofing, and other active attacks, making most domains and servers establish connections using HTTPS [21]. As of 2022, more than 79% of all websites on the Internet use HTTPS as the default protocol for communication [22]. Two methods are employed to force the use of HTTPS in web browsers which are:

1. **HTTPS Redirection:** HTTP redirects are used in web browsers in several scenarios. For instance, if a user wanted to access a website that has been previously updated or transformed to another new URL, the server responds with an HTTP status code 3xx to redirect to the new URL so the user can access the new domain [23]. Moreover, redirection is also used when a server that communicates using HTTPS transfers the communication from HTTP to HTTPS. For example, If a user wants to access a server that owns a valid SSL certificate and performs communication over HTTPS, by explicitly inputting HTTP in the URL, or by means of social engineering, a user is targeted to click an HTTP URL to initiate an attack, for example, http://www.example.com, the server sends a response back to the browser with an HTTP status code 301 (Internal Redirect), which is one form of HTTP 3xx redirection codes that indicates that the specified server is only accessible via HTTPS, hence forming an encrypted channel for secure communication between the browser and the server [24], as shown in figure 2.1.

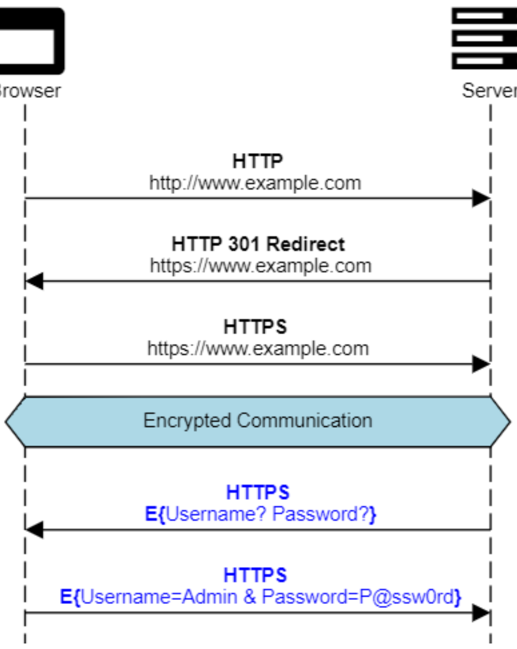


Figure 2.1: Internal HTTP to HTTPS Redirection

All browsers present to their users any issues regarding secure channel establishment, such as when the browser is unable to validate the server's SSL certificate due to its unknown issuer, mismatched domain names, if the certificate is expired, and so on, prompting the user to choose to continue interacting with the server in spite of the mentioned issues [25]. This practice is known as "*Click(ing) Through*" security [26]. However, browsers should not trust users as the main defense mechanism since oblivious users allow adversaries to exploit their illiteracy and lack of knowledge to break the most secure systems. It was surveyed that "*Click(ing) Through*" security stopped only 1 out of 300 users from proceeding to a banking website [27].

Despite the fact that HTTPS redirection provides security service, it still leaves room for an adversary to launch an SSL stripping attack to strip the SSL out of the communication and form an insecure channel as shown in figure 2.2. SSL stripping is a form of Man in The Middle (MiTM) attack that takes place when the user provides HTTP in the URL when communicating with a server that employs HTTPS. An attacker can eavesdrop and intercept the communication to prevent HTTPS redirection, by communicating with the server via HTTPS and with the user by HTTP. From this point forward, the adversary decrypts the communication received from the server and forwards it to the user in plaintext, and also forwards the plaintext data received from the user to the server in an encrypted manner [24], [28].

2. **HTTP Strict Transport Security (HSTS)** HSTS is documented in RFC 6797 from 2012 as a "mechanism enabling websites to declare themselves accessible only via secure connections and/or for users to be able to direct their user agent(s) to interact with given sites only over secure connections" [26]. In other words, HSTS can be interpreted as a security policy to mitigate SSL stripping attacks by forcing the employment of SSL in web browsers. This is declared by web servers using HTTP response headers.

   HSTS may operate between clients and servers either dynamically or statically. In dynamic operation, whenever an HSTS-enabled server is requested, it sends back a response header called the "Strict-Transport-Security" header with various parameters, such as the "max-age" that states the time for the browser to treat the domain as a trusted HSTS domain, to enforce the client to adhere to HTTPS communication. However, the

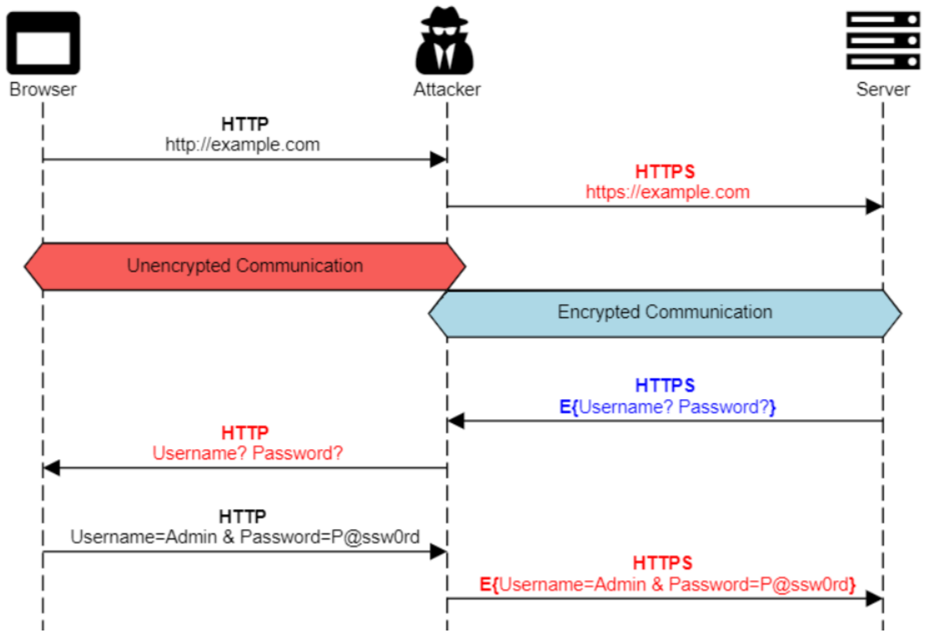


Figure 2.2: SSL Stripping Attack

HSTS header should be sent over HTTPS only and should be ignored by the browser if sent over HTTP. That being said, dynamic HSTS still addresses the *Trust-On-First-Use* (TOFU) issue [24].

On the contrary, static HSTS which is also known as “HSTS Preloading” defines a special list that is used and stored by most major browsers such as Chrome, Safari, Firefox, etc. that contains every domain that enables HSTS to be indicated as HTTPS only. Adding the domain to the HSTS preload list requires submitting a request via https://hstspreload.org. In this regard, browsers terminate HTTP requests before sending them to HSTS-enabled servers, and without prompting the user to the “*Click(ing) Through*” security [29], as shown in figure 2.3.

SMTP

The process of submitting a mail from a message user agent (MUA), e.g., Microsoft Outlook, to the email infrastructure, and relaying that mail to a trusted third party (from SMTP server to SMTP server; P2P), uses the *Simple Mail Transfer Protocol* (SMTP) [RFC2821] [30] that was designed in 1982 to transport and deliver emails over the Internet. However, using SMTP

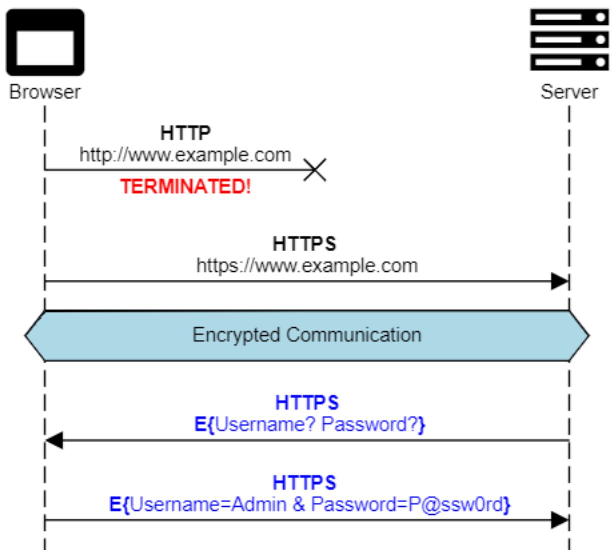


Figure 2.3: Internal HTTP Termination by Browser

standalone, communication was sent in clear text between clients and servers, leaving information unencrypted and endpoints unauthenticated. likewise, *Post Office Protocol* (POP3) and Internet Message Access Protocol (IMAP) which are responsible for retrieving emails from the server to the computer also operate without confidentiality and authenticity [31]. That being said, the development of secure SMTP, POP3, and IMAP protocols were introduced in early 1997 by the IANA as the *Implicit TLS* by modern standards [32]. Implicit TLS is defined to be used over port 465 for SMTP, and ports 995 and 993 for POP3 and IMAP, respectively [31]. It secures the connection by starting a TLS handshake and then exchanging all commands and communication as encrypted TLS data. However, since this was not submitted to the IETF as an RFC, port 465 was used for another application and Implicit TLS has been deprecated for SMTP [33]. Against this background, the IETF introduced *STARTTLS*, which is a protocol command that announces starting a TLS session using port 587 [34], as illustrated in figure 2.4.

The process starts with the *Transmission Control Protocol* (TCP) handshake between the client and server to identify each other. After which the client issues the EHLO command to obtain the server's capabilities. After the server announces its TLS capability (250 STARTTLS), the client gives the STARTTLS command, waiting for the code 220 (server is ready to start TLS), hence TLS handshake begins. However, if the client receives a code 454 (TLS is not currently available) or if certificate validation fails, the client decides whether to abort the connection

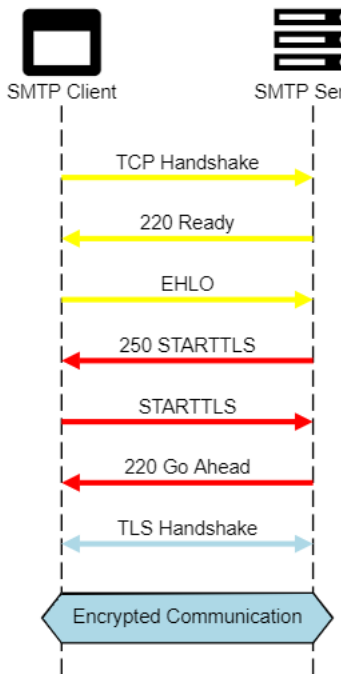


Figure 2.4: Securing SMTP Messages With STARTTLS

or fall back to send the message in plaintext by showing users TLS exceptions and prompting them to decide.

The usage of STARTTLS introduces drawbacks and vulnerabilities. First, STARTTLS undoubtedly adds some latency to the SMTP connection process due to its negotiation process. Moreover, since the initial TCP connection is unencrypted, IP addresses are sent on the clear, hence allowing adversaries to initiate spoofing and command injection attacks. additionally, a STARTTLS stripping attack can be launched by intercepting the communication and preventing the server to send the response code 220, this can easily downgrade the communication to plaintext [31].

According to a survey performed by *Chan et al.,* in [33] after analyzing 681GB of packet capture files that correspond to HTTP, SMTP, IMAP, POP3, and FTP, from 2008 to 2017, results illustrated that STARTTLS is significantly more applied than the Implicit TLS for SMTP traffic, as traffic that used STARTTLS option was approximately 20 times more than the Implicit TLS variant. However, the IETF states that “It is desirable to migrate core protocols used by MUA software to Implicit TLS over time” [34].

TCP

Despite the fact that SSL/TLS is being progressively adopted in almost all data transmission, a considerable portion of TCP traffic remains unencrypted. Hence, the development of *TCPCrypt* was mandatory to ensure the confidentiality of exchanged communication through the internet. TCPCrypt is a TCP extension intended to accomplish end-to-end encrypted channel between any two applications that use TCP as their communication protocol (Client-Server, P2P). Figure 2.5 illustrates the TCPCrypt mechanism [35].

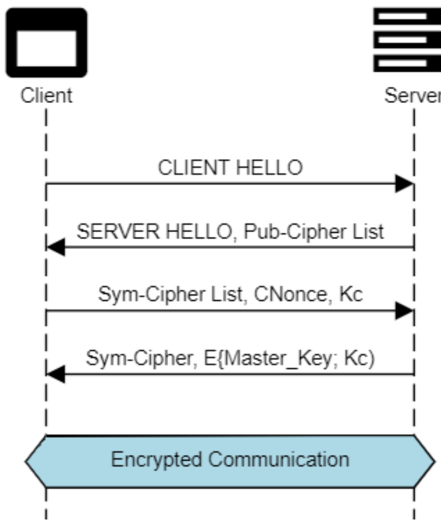

Client
Server
CLIENT HELLO
SERVER HELLO, Pub-Cipher List
Sym-Cipher List, CNonce, Kc
Sym-Cipher, E{Master_Key; Kc)
Encrypted Communication

Figure 2.5: TCPCrypt Connection Establishment

The process starts when the client initiates a TCP connection and requests the server to encrypt the upcoming communication. Similar to SSL/TLS, the server accepts by providing its public-cipher suite, then the client chooses an algorithm, presents its symmetric-cipher suite, and sends a random nonce and its public key. Finally, the server encrypts the master key used for upcoming encryption using the client's public key and forwards it to the client, allowing all further communication to be encrypted.

A key benefit of TCPCrypt, it does not require any configuration or changes to applications to perform encryption [35]. However, if one endpoint does not support TCPCrypt, all communication will fall back to standard plaintext TCP, which is referred to as *Opportunistic Encryption*, which makes TCPCrypt vulnerable against MiTM. Moreover, Standard TCPCrypt does not guarantee authenticity for communication between endpoints, as it typically does not involve X.509 certificates or passwords, as opposed to SSL/TLS [36].

DHCP

The first protocol a new host invokes when connecting to a network is *Dynamic Host Configuration Protocol* (DHCP), which supports new clients with network configuration parameters (subnet mask, IP address, gateway IP address, etc.). Although DHCP is one of the most used client-server protocols in the networking realm, it possess a significant security risk as DHCP communication is sent and received in plain text and does not involve any authentication mechanism which contributes to performing the so-called "starvation/spoofing attack". It occurs when an insider launches a DoS attack targeting the DHCP server to exhaust the IP address pool. Hence, DHCP messages from new clients intending to join the network will be transferred to an adversary that acts as a DHCP server. The attacker assigns the new client with a legitimate IP address; however, the rogue server also assigns its IP address as the gateway. Thus, any communication sent from the victim is intercepted and sniffed by the attacker [37]. Although DHCP Snooping prevents starvation attacks it still opens the floor for other kinds of attacks since it doesn't achieve confidentiality and authenticity as shown in [38].

DHCP Snooping is the recognized and documented defense mechanism against starvation attacks, it classifies interfaces/ports into trusted and untrusted. If any DHCP message normally sent by a server is received through an untrusted port will be discarded, and other messages that are normally sent by clients are filtered based on their DHCP MAC addresses, Ethernet header MAC addresses, and IP addresses. Moreover, DHCP Snooping makes use of a binding table to register IP and MAC addresses for all the traffic the switch sees to make filtering decisions [39].

To epitomize the mentioned networking protocols and the standard security mechanism each protocol implements to mitigate attacks and secure communication, Table 2.1 compares if the security mechanism applies for P2P communication, performs encryption, based on SSL to perform authentication and encryption, and if it enforces encryption or terminates the communication when one party is not configured with the same mechanism.

Table 2.1: Comparison Between Security Mechanisms of Mentioned Networking Protocols

Protocol	Security Mechanism	Peer-to-Peer	Achieves Encryption	SSL-Based	Forces Encryption
HTTP	HTTPS	Yes	Yes	Yes	No
	HTTPS/HSTS	No	Yes	Yes	Yes
SMTP	STARTTLS	Yes	Yes	Yes	No
TCP	TCPCrypt	Yes	Yes	No	No
DHCP	DHCP Snooping	No	No	-	-

2.3 Current Solutions

While we explore the challenges associated with the established networking protocols and their security measures, it becomes evident that the ever-evolving landscape of secure communication calls for innovative approaches. However, although a bulk of research has been concentrated on the development of load balancing techniques and algorithms to enhance their performance and reliability [40], research topics that consider encrypting the back-end communication channels are still in infancy. Moreover, research papers that focused on improving the performance of intra-domain routing protocols and fixing their instabilities lacked the inclusion of solutions for unencrypted and unauthenticated communication between end-points in intra-domain environments [41].

One approach that addressed the issue of intra-domain communication and developed an SSL-based framework to improve the authenticity of DHCP communication was proposed by *Shue et. al.,* in [8]. The framework relies on the configuration of a public-key pair for the local organization's domain and servers and utilizes the domain's private key to sign generated server certificates to prove authenticity, each of which contains each server's public key. Then, The DHCP server is used to issue users' certificates after verifying their authenticity using a captive portal to enter their credentials. Their approach successfully achieved endpoints authenticity as their goal was to mitigate DHCP spoofing attacks. However, their approach and authentication procedure did not specify any key exchange algorithm for encrypting subsequent communication. Moreover, certificate management practices were not implemented, such as certificate revocation, validation, and renewal.

Another new and recent approach was implemented by Microsoft for SharePoint server-to-server authentication in a SharePoint farm. SharePoint administrator creates a certificate for each SharePoint server, submits it to a third-party certificate authority to generate a signed certificate, and then imports it back to the Windows Certificate Store on each SharePoint server. Hence, whenever a SharePoint server attempts to claim information from its peer, the sent request must be accompanied by the sender's certificate to initiate an SSL handshake and then encrypt further communication [42]. Unlike the prior approach, SSL handshake and key-exchange algorithm are implemented, plus, all certificate management operations are considered and executed by a certificate authority. However, managing certificates for each SharePoint server individually can become complex, especially in larger farms with multiple servers.

A different method involved *Module-OT* which is an HSM developed by the *National Renewable Energy Laboratory* (NREL) to secure communication between distributed energy resource systems. *Module-OT* is one of the various HSMs developed by different vendors, where each of which has its own specifications and features. *Module-OT* also complies with the (FIPS) 140-2 standard [18]. They tested their model by placing two HSMs to take control of data encryption and decryption between a data generator and a data emulator using symmetric-key cryptography, without performing authentication and SSL encryption [43].

2.4 Discussion

Based on studying current networking protocols and their security structures in table 2.1, and investigating the existing solutions developed by academic researchers and industry, it is evident that no mechanism provides optimum security for confidential and authenticated communication, and incorporating them for the goal of securing communication leads to security holes. Plus, current solutions only apply to specific types of environments or servers, and do not utilize all cryptographic operations for robust security.

Thus, a uniform security mechanism is required to be employed in intra-domain environments, that integrates SSL in conjunction with all other networking protocols to secure web-based and non-web-based communication. However, synthesizing SSL requires a numerous number of cryptographic keys and certificate management practices. The common approach to solve the aforementioned case is the usage of *Key Management Service* (KMS), which is software installed in servers or the cloud, that is responsible for helping organizations in managing cryptographic keys and certificates securely. One example of a cloud-based KMS is Amazon's *AWS KMS* [44]. Amazon's KMS makes it easier for organizations to generate, store, manage, and distribute keys. Plus, it acts as a certificate authority to perform all certificate management practices.

HSMs on the other hand are physical devices installed within organizations to perform equivalent functions to the KMS systems. However, the main key differences between the two include:

- *Security*: HSMs are highly secure devices as they conform to the FIPS 140-2 standard, thus making them provide more reliable than their competitor. Whereas a vulnerability in Amazon's KMS has been exploited recently by *Thai Duong* [45] that caused client's private information to get leaked, among other risks.

- *Flexibility*: KMS systems are usually easy to be integrated with software programs and services in servers. However, HSMs require vendor-specific libraries to operate. This greatly reduces the flexibility when an HSM is substituted with another from a different vendor.

- *Cost*: KMS systems are pay-as-you-go services, and are often proving to be a cost-effective alternative compared to the deployment of HSMs, which typically demand up-front cost or long-term commitments.

Compared to software-based alternatives, the verifiable security provided by HSMs induces the need to employ HSM services in intra-domain environments. However, their vendor-specific libraries can make deployment burdensome when switching between HSMs. To overcome the associated flexibility and cost issues with HSMs, we propose developing the *SSL Everywhere* architecture.

SSL Everywhere is a *Software as a Service* (SaaS) architecture designed to revolutionize the way organizations implement secure communication within intra-domain environments, as depicted in Figure 2.6. This innovative approach combines the power of HSMs with the flexibility and scalability of cloud-based services. SSL Everywhere provides a standardized way to manage and secure intra-domain communication within organizations. By integrating industry-standard cryptographic protocols, including SSL, and leveraging the capabilities of HSMs, SSL Everywhere ensures the confidentiality, integrity, and authenticity of data exchanged between endpoints. Figure 2.6 shows the SSL Everywhere service model, where two domains in two autonomous systems within the same organization, Domain A and Domain B, are subscribed to the SSL Everywhere services, both utilize the SSL Everywhere service and have access to a suite of security and SSL certificate management tools.

Moreover, this architecture offers a zero-touch deployment model, reducing the time and effort required for implementation, and allowing organizations to seamlessly integrate SSL Everywhere into their existing infrastructures with minimal configuration. SSL Everywhere is dedicated to securely generating, storing, and managing cryptographic keys and certificates, as it offers standardized key management using a user-friendly command-line tool. A more detailed explanation of the architecture is explained in Chapter 3.

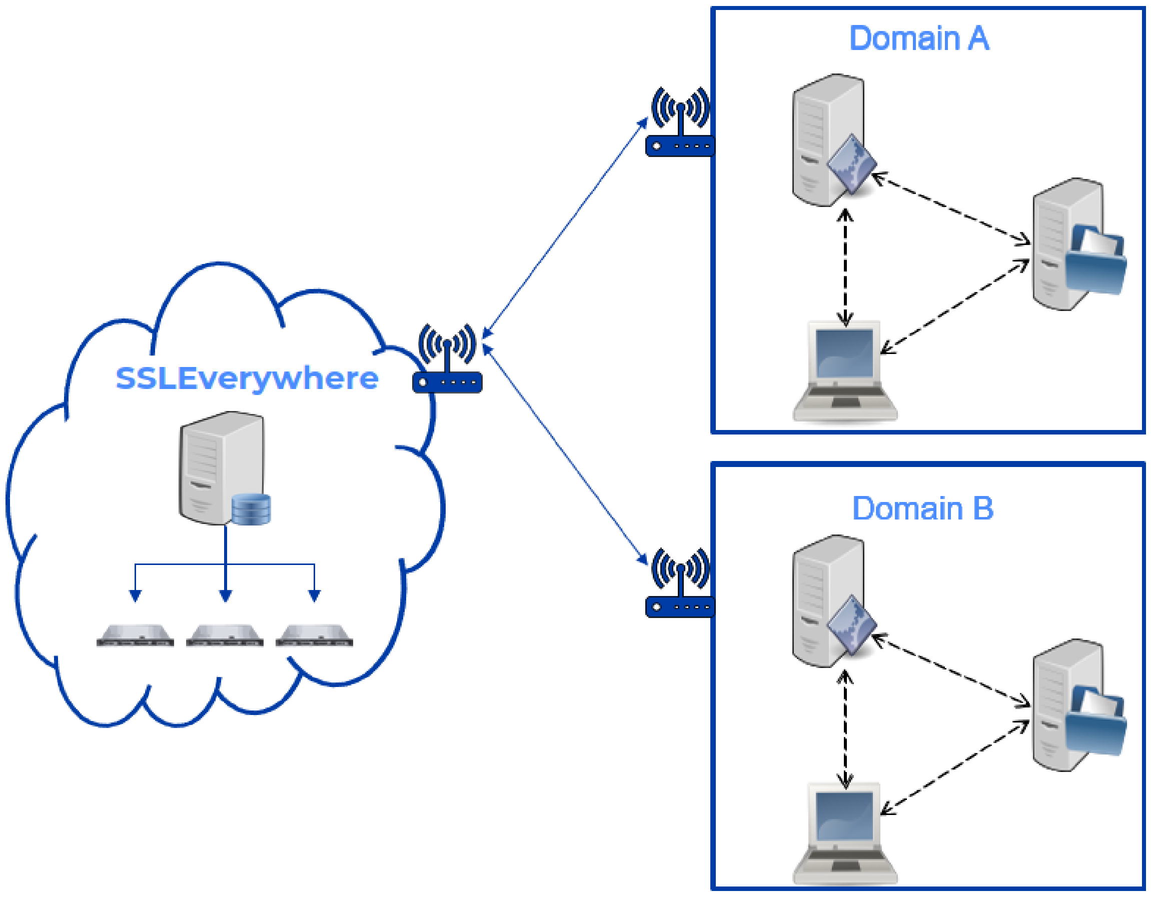


Figure 2.6: SSL Everywhere Service Model

Chapter 3

SSL Everywhere Architecture

This chapter proposes a comprehensive architecture that provides seamless integration of various HSMs that collectively fortify the confidentiality, integrity, and availability of sensitive data transmitted across intra-domain networks. The purpose of the SSL Everywhere architecture is to enhance the security and performance of communication systems that utilize the SSL protocol within intra-domain environments by providing various *security services* based on HSMs' capabilities. Some of the key security services offered by SSL Everywhere include:

1. Secure Storage Initialization.
2. Key-Pair Generation
3. Digital Certificate Generation
4. Data Encryption/Decryption
5. Access Control and Authorization

By utilizing HSMs capabilities that employ advanced encryption algorithms, secure key and digital certificate management practices, stringent access control, and providing unified access to their services, SSL Everywhere helps protect intra-domain environments from unauthorized access, manipulation, and interception.

3.1 General Architecture

Starting with the SSL Everywhere general architecture, shown in Figure 3.1, it is a 3-layer architecture designed to facilitate unified and uniform access from software applications to various HSMs.

At its foundation, the architecture features the physical layer, which includes specialized HSMs from multiple vendors, where communication between the HSMs and the upper layers is facilitated by the *Adaptive Driver Communication Protocol* (ADCP). These HSMs provide secure storage for cryptographic keys and certificates, perform cryptographic operations, and enforce robust security measures.

The middle layer of the architecture is the management layer, which consists of several *service functions* and the *Hardware Abstraction Layer* (HAL). The service functions encompass multiple managers, each dedicated to a specific cryptographic operation or the management of the SSL Everywhere. The HAL acts as an interface between the physical layer and the higher-level components to facilitate unified communication by abstracting the differences between the underlying HSMs.

Finally, the top layer is the application layer, where SSL protocol and applications reside. This layer leverages the underlying hardware and software components through an *Application Programming Interface* (API), to establish secure and reliable communication channels.

The significance of SSL Everywhere is that the management layer will provide a uniform view of HSM services so that API functions can be implemented independently of the specific hardware of the vendor. The APIs set uniform interaction methods that act as the main gateway to expose bidirectional REST protocols to be used for the applications. These applications are not required to run in the same domain space or even on the same machine. Another significant aspect of this prototype design is that it is not tied to one HSM. This provides greater flexibility in constructing adaptive KMS. The key element of the management layer is to abstract the complexity of the HSMs' configurations into a set of features that can be invoked by the applications via uniform APIs.

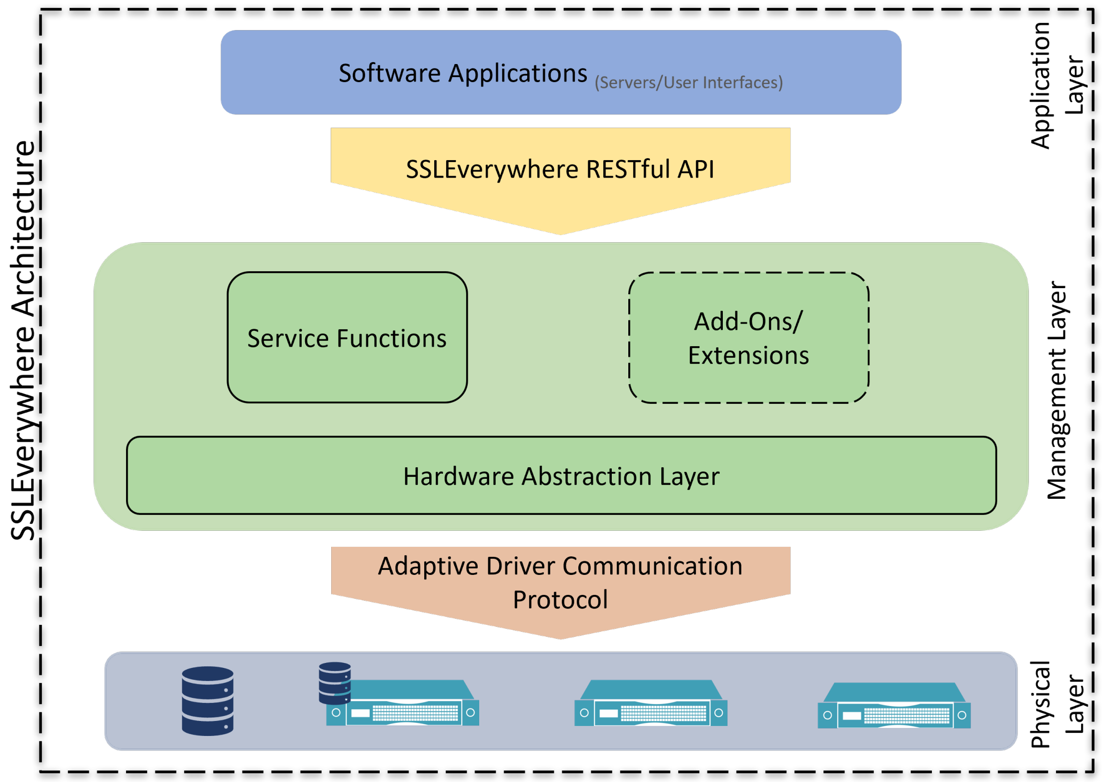

SSLEverywhere Architecture
Software Applications (Servers/User Interfaces)
Application Layer
SSLEverywhere RESTful API
Service Functions
Add-Ons/ Extensions
Hardware Abstraction Layer
Management Layer
Adaptive Driver Communication Protocol
Physical Layer

Figure 3.1: SSL Everywhere High-Level Architecture

3.2 Low-Level Architecture

Now that we have established an overview of the SSL Everywhere, the low-level architecture of the SSL Everywhere, shown in Figure 3.2, offers an in-depth exploration of the internal components and their intricate interactions within the system. This section provides a detailed description of each layer, highlighting their specific functionalities and contributions to the overall architecture.

3.2.1 SSL Everywhere Application Layer

Starting with the application layer, it is the uppermost layer that consists of software applications and user interfaces running on servers that utilize cryptographic services provided by the HSMs through the SSL Everywhere architecture. The layer interacts with the management layer through an API to provide access to different HSMs, enabling seamless access to the various functionalities and capabilities of the underlying HSMs.

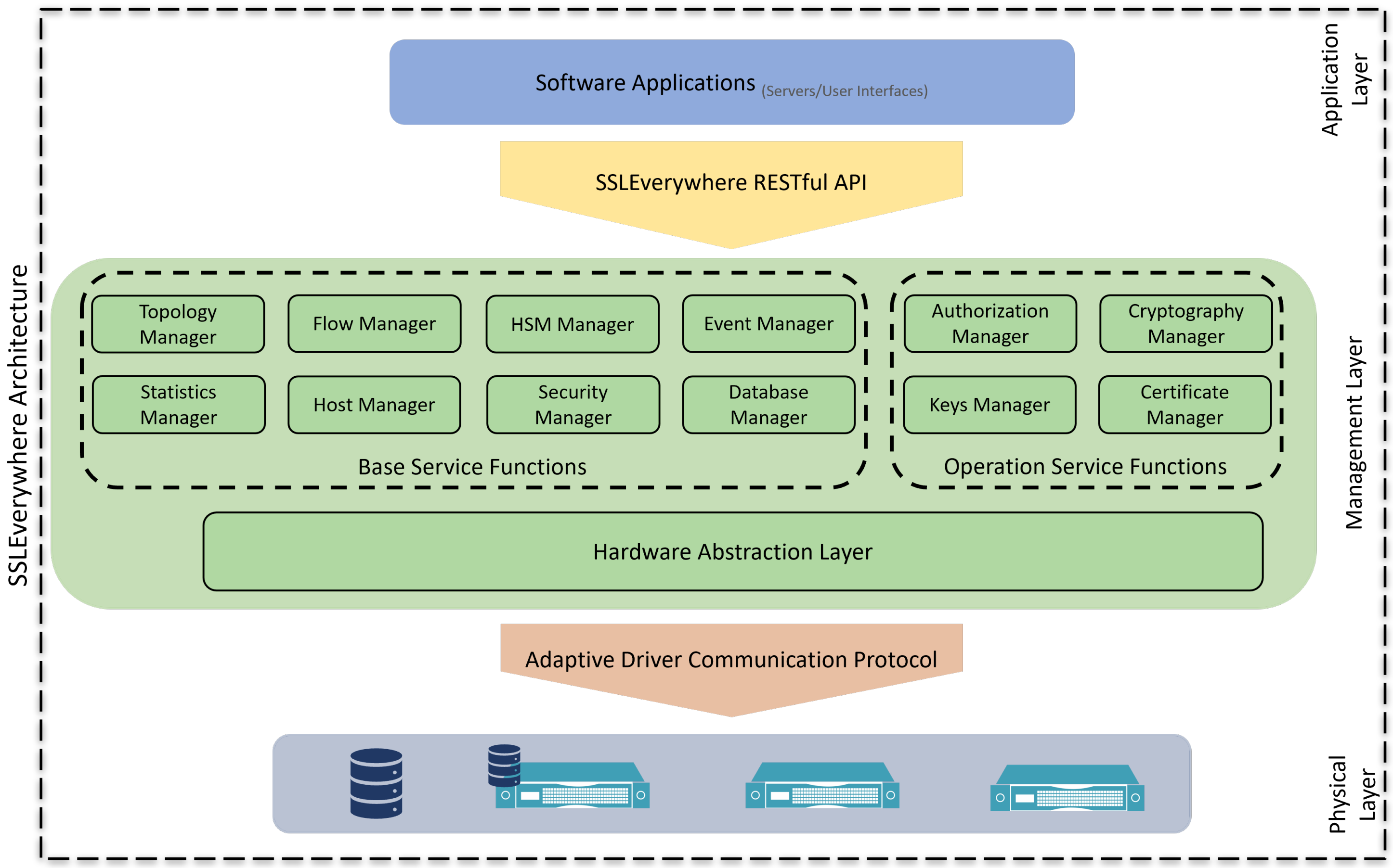


Figure 3.2: SSL Everywhere Low-Level Architecture

3.2.2 SSL Everywhere API

To allow applications in the application layer to communicate with the management layer, various frameworks can be employed to develop RESTful APIs, that follow the REST architecture by using HTTP requests to communicate with resources identified by URLs, and exchange information through the standard HTTP methods, including GET, POST, PUT, and DELETE. Some of the most widely used and recognized frameworks include:

- **Flask**, developed in 2010, is a micro web-framework to create APIs using Python. It only provides the core functionality that is necessary to build an API, leaving the rest of the extra features to be added by the developers as required, making Flask a very simple and easy-to-use API development tool. Additionally, whether building a basic application or a full-featured API, Flask offers comprehensive documentation full of examples and implementation details. The challenge of using Flask as a micro-framework is that developers have to take extra steps to secure the application. This can be a source of concern if not properly managed, thus making it less suitable for large-scale applications that require advanced functionalities.

- **Django**, debuted in 2005, is a full-featured Python framework. With more than 2500 packages included in its library, it prioritizes the security of applications by guarding against various types of attacks. This reduces the burden for developers to configure applications' security by employing third-party extensions. Another advantage of using Django is the extensive amount of documentation and tutorials due to its prolonged history of operation.

- **FastAPI** is a high-performance micro-web framework designed for building APIs with Python. It was first released in 2018. It provides an easy-to-use and intuitive syntax that simplifies the development process while ensuring optimal performance. FASTAPI includes automatic generation of API documentation, a simplified user interface to test the API, and data validation, among other features.

- **Spring** is one of the longest-standing Java-based API development frameworks. Initially released in 2002, it is widely adopted for developing enterprise-level applications, including RESTful APIs. Over the years, Spring has evolved into a full-featured ecosystem of libraries and extensions that can be used to develop a wide range of applications. Its popularity and extensive documentation are a testament to its stability, robustness, and flexibility, making it a go-to choice for developers in the Java ecosystem.

In Chapter 4, we will provide a comparison of the various API development tools that are presented based on their performance. Based on the comparison results, a selected toll will be identified for integration with the SSL Everywhere.

3.2.3 SSL Everywhere Management Layer

Now, as we shift our focus from the API, we'll navigate towards the management layer, where SSL Everywhere's centralized control and monitoring mechanisms come into play. Figure 3.2 shows the detailed architecture of the management layer, which exposes the higher-level APIs to communicate with the underlying HSMs. It is primarily made up of multiple service managers who can be classified as follows:

Base Service Functions

The base service functions are employed to provide the primary services required to function the SSL Everywhere properly. Base service functions include:

- **Topology Manager:** Stores information about the HSMs and organizes HSMs with their physical and logical connections in a hierarchical structure. The topology manager monitors newly added and removed HSMs to construct this structure dynamically.

- **Statistics Manager:** Collects statistical information from the underlying HSMs, such as request counts and error rates. It stores and maintains this information to comprehensively generate reports based on the collected statistics.

- **Flow Manager:** Creates policy sets that guide the request routing process. These policy sets define the rules and criteria for determining the appropriate destination for each incoming request. By configuring these policies, the flow manager guides the HAL, enabling it to determine the specific HSM to which a request should be sent. These policies can consider various factors, such as security requirements, load balancing, the availability of HSM resources, and other relevant considerations. It also directs requests to the appropriate HSM based on pre-defined flow policies.

- **Host Manager:** Manages and stores connected end-point hosts' information. It maintains a repository that stores crucial details about each host, including their MAC addresses and IP addresses.

- **HSM Manager:** Manages and stores the underlying HSMs' information. It maintains a centralized repository of HSM-related data, including their statuses and configurations. This information includes details about the available HSMs, their connectivity, operational status, and any relevant configuration settings.

- **Security Manager:** Analyzes the HSMs' states and incoming data to identify anomalies. Machine-learning-based anomaly detection systems can be employed in the security manager to continuously scan incoming requests and detect threats.
- **Event Manager:** Collaborates closely with the security manager. Its primary role is to monitor critical events within the HSMs. By continuously monitoring various aspects of the system, such as performance, and security indicators, in any critical event, such as a security breach or system failure, the Event Manager generates reports and alerts.
- **Database Manager:** Manages and backups data storage, and generates statistics about saved data, such as certificates and audit logs.

Operation Service Functions

On the other hand, the operation service functions contain managers oriented to perform specific cryptographic and HSM-related tasks, such as:

- **Authorization Manager:** Performs host authentication, and authorization, and determines which hosts can access HSM resources. It is responsible for validating the identity and credentials of connecting hosts and determining their level of access to HSM resources. By enforcing access control policies and rules, the Authorization Manager ensures that only authorized hosts can utilize the HSM resources and perform cryptographic operations.
- **Cryptography Manager:** Performs cryptographic operations, such as encryption, and decryption, and secure storage initialization within the HSMs to store cryptographic objects.
- **Keys Manager:** Handles key generation and retrieves cryptographic keys for hosts, for encryption and decryption processes.

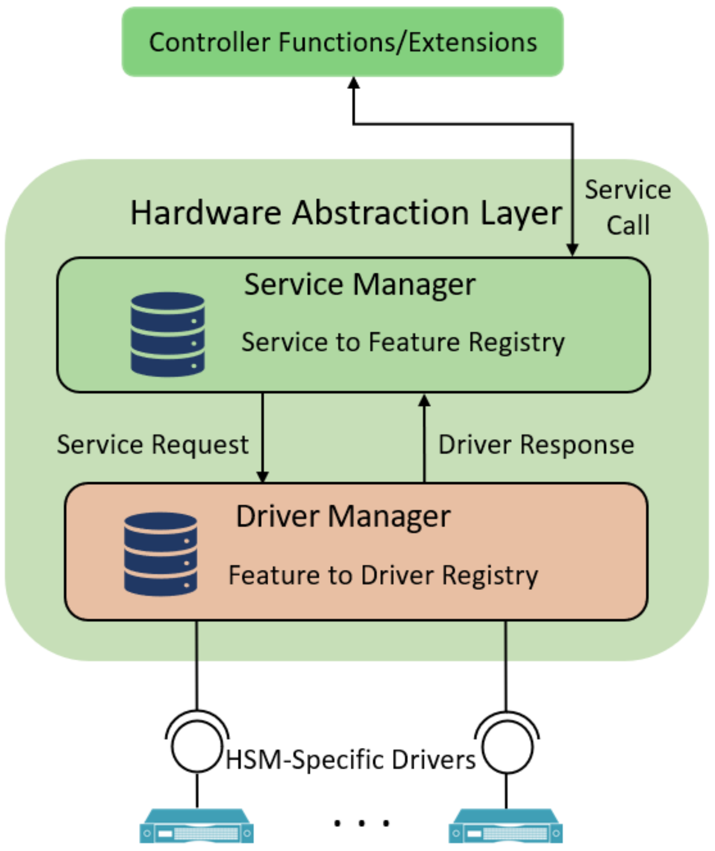

Controller Functions/Extensions
Hardware Abstraction Layer
Service Call
Service Manager
Service to Feature Registry
Service Request
Driver Response
Driver Manager
Feature to Driver Registry
HSM-Specific Drivers

Figure 3.3: Hardware Abstraction Layer

- **Certificate Manager:** Manages the validation and revocation of digital certificates used for secure communication and authentication purposes. It maintains a repository of trusted root certificates and intermediate certificates, enabling the validation of certificates within the system. The Certificate Manager also handles the revocation of certificates in case of compromised or expired certificates

Hardware Abstraction Layer

Moving beyond the management layer, we'll now delve into the hardware abstraction layer. The HAL, shown in figure 3.3, is the heart of the SSL Everywhere. It enables software applications to communicate with different HSMs, by abstracting the low-level details of the underlying HSMs. It contains a *Service Manager*, which is responsible for receiving a request from the controller managers and translating it into a feature request based on a predefined policy set, then forwarding it to the *Driver Manager*, which has a registry that maps each feature with the appropriate HSM's driver command that handles the request.

3.2.4 Adaptive Driver Communication Protocol (ADCP)

The aforementioned SSL Everywhere components need to communicate with each other, this communication is facilitated using the ADCP. The ADCP is a JSON-like messaging protocol that forms the backbone of the communication infrastructure within the SSL Everywhere by managing communication between the applications and the HSMs' drivers. It provides a standard format for requests and responses to ensure consistent communication among various HSMs.

3.2.5 SSL Everywhere Physical Layer

The physical layer of the SSL Everywhere architecture encompasses the hardware components and infrastructure necessary for the secure operation of the system. This layer plays a critical role in ensuring the confidentiality, integrity, and availability of sensitive data and cryptographic material. It encompasses the following components:

- **HSMs**: The physical layer includes tamper-resistant HSMs that provide cryptographic services. These HSMs are designed to safeguard sensitive data and cryptographic keys.
- **Databases**: For supplementary storage capacity, the physical layer may include databases that store cryptographic elements, such as keys, certificates, and audit logs. These databases can be located either on the same server as the HSMs or on a separate server.
- **Simulators**: In addition to physical HSMs, the physical layer may also include software HSM simulators, also known as virtual HSM VHSM. These simulators, such as SoftHSM [46], replicate the functionalities of physical HSMs in a software-based environment. They are often used as a playground for training, development, and testing purposes, providing a cost-effective and flexible alternative to physical HSMs.

Simulator

It is important to note that the SSL Everywhere architecture has been developed and thoroughly tested on the SoftHSM simulator environment. SoftHSM provides a software-based emulation of an HSM and offers a controlled environment for cryptographic operations and secure storage. Throughout this research, the SoftHSM simulator has served as a reliable and flexible tool for assessing the architecture's performance and providing cryptographic services to clients.

Database

The SSL Everywhere database serves as the backbone of the architecture, storing and managing critical information and operational data. Within this robust database, three key tables play pivotal roles in tracking, recording, and organizing essential components and transactions. These tables include the "HSMs" table, the "Hosts" table, and the "Certificates" table. Each table serves a unique purpose, collectively contributing to the seamless operation and security of the SSL Everywhere architecture.

The "HSMs" table in Table 3.1 serves as a comprehensive repository of HSMs information, providing essential insights into the management and performance of these components. Each record in the table is uniquely identified by an *"ID"* assigned to every HSM instance, facilitating efficient data retrieval and management. Additionally, the *"Name"* column allows users to assign human-readable identifiers to each HSM, aiding in easy reference and recognition.

One important aspect of HSM operation is captured by the "Initialized" column, which indicates whether an HSM has been successfully initialized and configured for use Furthermore, the *"Processed Requests"* column quantifies the number of cryptographic requests or operations carried out by each HSM since its initialization. This metric not only helps in monitoring the workload of individual HSMs but also aids in resource allocation and optimization.

Lastly, the *"Status"* column provides real-time information regarding the operational status of each HSM, with possible values including "Active", "Offline", or "Error." The status tracking feature offers valuable data for proactive maintenance and troubleshooting, ensuring the continued reliability and security of the HSM infrastructure.

Moving on, the *"Hosts"* table in Table 3.2 within the SSL Everywhere database serves as a repository of information regarding the networked hosts that interact with the architecture. With well-defined columns such as *"ID"*, *"MAC Address"*, *"IP Address"*, *"Requests Count"*,

Table 3.1: HSMs Information Table

Column	**Type**	**Description**
ID	Int	Unique HSM identifier for each record in the dataset
Name	String	A user-defined string that serves as a human-readable identifier for each HSM.
Initialized	Boolean	Indicates whether the HSM has been successfully initialized and configured for use.
Processed Requests	Int	Represents the number of cryptographic requests or operations processed by each HSM since its initialization.
Status	Boolean	Reflects the operational status of each HSM (Active/Offline/Error).

Table 3.2: Hosts Information Table

Column	**Type**	**Description**
ID	Int	Unique identifier for each host in the table.
MAC Address	String	The unique MAC address of the host.
IP Address	String	The IP address assigned to the host.
Requests Count	Int	Represents the total count of requests made by the host.
Admin Email	String	The email address of the administrator or contact person responsible for the host's management and maintenance.

and *"Admin Email"*, this table efficiently captures and manages key details about each host. The *"ID"* column provides a unique identifier for every host recorded in the table.

Meanwhile, the *"MAC Address"* and *"IP Address"* columns store the host's unique hardware and network addresses, facilitating precise identification and routing. The *"Requests Count"* column keeps track of the total number of requests initiated by each host, providing valuable insights into their communication patterns. Lastly, the *"Admin Email"* column captures the email address of the individual responsible for managing and maintaining the host, this information is also used when creating the digital certificate for the user.

Table 3.3: Certificates Information Table

Column	**Type**	**Description**
ID	Int	Unique identifier for each certificate in the table.
MAC Address	String	The MAC address associated with the host for which the certificate is issued.
Certificate	BLOB	The binary data representing the actual certificate issued to the host.
Certificate Chain	BLOB	The binary data representing the certificate chain associated with the certificate.

Further, Table 3.3 shows the *"Certificates"* table within the SSL Everywhere database is involved in securely managing and storing cryptographic certificates used in the secure communication architecture. The *"ID"* column serves as a unique identifier for each certificate record. The *"MAC Address"* column associates each certificate with the specific host for which it is issued, creating a direct link between certificates and their respective hosts. The *"Certificate"* column stores the actual certificate in binary form. Simultaneously, the *"Certificate Chain"* column captures the binary data representing the associated certificate chain, this is crucial for validating the authenticity of the certificate. This table's particular organization of certificate-related information ensures the reliable operation of SSL Everywhere.

Chapter 4

SSL Everywhere Design Process & Methodology

In the SSL Everywhere design process, we employ a bottom-up approach that progresses from the physical layer to the application layer. This sequential methodology allows us to establish a solid foundation by addressing the fundamental aspects of the system architecture before proceeding to higher-level functionalities. By starting at the physical layer, we ensure that HSMs' services are well-defined and expressed in command lines. This fosters the construction of a standard JSON message to be applied to all security services provided by SSL Everywhere.

Once we ensure that the underlying infrastructure, including the HSMs, secure storage, and communication protocols, is well-defined and effectively integrated, we proceed to construct the HAL and its components.

Subsequently, we proceed to progressively expand upon this groundwork, methodically integrating the essential elements and capabilities of the application and management layers.

4.1 ADCP Standard Message

In what follows, we start by describing the JSON-based standard message format that serves as a robust communication template for interacting with the HSMs, starting from the physical layer. This standardized format shown in Figure 4.1 provides a structured and comprehensive approach to exchanging information and instructions, ensuring consistent integration across diverse systems. By adhering to this format, organizations can establish a common language for HSM interactions, facilitating secure and efficient communication.

The process behind designing the standard JSON message involved studying the documentation and services of the available HSMs. By thoroughly examining the capabilities and specifications of various HSMs, we were able to design a format that captures the essential parameters and functionalities required for proper communication with HSMs, where each specific command corresponds to a JSON message.

In this standard message format, the message is structured into two main sections: the header and the body. The header section encompasses essential information that is consistently present in all messages, irrespective of their specific purpose. It serves as a meta-information container, providing crucial context and facilitating the proper routing and interpretation of the message.

The header includes parameters such as *message_direction* which denotes if this message is a request for a service or a response to the client. This parameter can also hold a value to indicate that a message is for internal purposes, such as messages between managers for information exchange. Another parameter is the *"message_id"*, which assigns a unique serial number to each message, starting from "1" and incrementing with each new message. This enables the tracking and referencing of messages within the system. The *"message_type"* parameter indicates the specific HSM operation to be performed. The *"source_mac_address"* parameter represents the MAC address of the device or entity sending the message, enabling internal architecture usage and the association of cryptographic elements with their corresponding hosts. Similarly, the *"source_ip_address"* parameter denotes the IP address of the source device, providing additional internal architecture usage information. Further, the *"timestamp"* parameter in the header serves to record the time at which the message is generated, providing a temporal reference for tracking and synchronization purposes within the architecture.

Additionally, the header contains the *"dest_hsm_module"* parameter, which specifies the destination HSM module where the command should be executed. It identifies the path to the HSM module library file responsible for handling the requested operation. These header parameters collectively establish the necessary foundation for message routing and proper interaction with the HSM. By incorporating these standardized header parameters, the message format ensures consistency, interoperability, and efficient processing across different HSM-related operations and systems. The body section of the message format, on the other hand, encapsulates the specific content and parameters related to the intended operation or task. It holds information that is specific to the particular message and varies based on the desired functionality.

Within the body section, additional parameters and details that are specific to the operation defined in the header are included. For instance, the *"session_label"* parameter represents the label or name assigned to the secure session being initialized, or an identification on where to store generated cryptographic elements. This allows for proper identification and management of different sessions within the HSM.

The *"credentials"* object contains the *"admin_pin"* and *"user_pin"* parameters. These credentials are utilized for administrative and user-level operations, granting access to the token's cryptographic functions and enabling authentication and authorization. Furthermore, the *"key_info"* object encompasses parameters related to the generation of cryptographic key pairs. The *"type"* parameter specifies the type of cryptographic algorithm to be used, such as RSA, AES, or EC. The *"length"* parameter denotes the desired length of the generated key. Additionally, the *"label"* parameter assigns a unique name or label to the generated key pair, facilitating identification and management. Finally, the *"id"* parameter provides an id number for the generated key pair, enabling easy referencing and retrieval when needed.

Moreover, for X.509 certificate generation requests, the *"csr_details"* component serves as a repository for storing critical CSR details. These details encompass two key attributes: *"csr"* which encapsulates the CSR data received from the client for signing, and *"csr_file"* which specifies the file path where the CSR is to be saved within the SSL Everywhere server. This streamlined approach of saving the CSR file before proceeding with the signing process significantly streamlines the certificate generation process. Along with the *"csr_details"*, the second component, *"cert_details"* is also involved in the certificate generation security service. *"cert_details"* namely includes two important parameters. First is the *hash_algorithm* which denotes the hashing algorithm to be used for signing the certificate. If not specified by the user, the *hash_algorithm* is *"sha256"* by default. The second is *"days"* which specifies the certificate validity in days, which is hard-coded *"365"* days and can be changed by the SSL Everywhere administrators.

By utilizing the standard ADCP message as the foundational means of communication within the SSL Everywhere framework, security service requests can be transmitted across the infrastructure with optimal efficiency.

4.2 Security Services

The ADCP message plays a crucial role in providing various security services to clients. SSL Everywhere security services provide a wide range of cryptographic operations and HSM-related functions. Those services play a pivotal role in safeguarding sensitive information and ensuring the integrity and confidentiality of cryptographic assets. With SSL Everywhere's security service capabilities, they provide a solid foundation for implementing secure systems and maintaining the confidentiality and integrity of exchanged data in intra-domain environments. These services include, but are not limited to:

4.2.1 Secure Storage Initialization:

The first security service is described by the JSON message shown in Figure 4.2 that facilitates secure storage initialization in an HSM. Initializing secure storage is a critical initial step in leveraging HSMs, as it creates a secure and controlled environment within the HSM for storing and managing cryptographic elements, such as keys and certificates.

The header section contains information such as the message ID, message type (initsession), source MAC address, source IP address, and destination HSM module. The body section includes the session label, which is set to (SecureSession1) for storage identification and management purposes. The credentials object includes the admin PIN and user PIN. The *"admin_pin"* provides administrative privileges for managing the token and performing privileged

```
{
        "header":{
        "message_direction": "request/response",
        "message_id": "msg_id",
        "message_type": "requested_operation",
        "source_mac_address": "src_mac",
        "source_ip_address": "src_ip",
        "timestamp": "2023-11-14T16:00:36.937948",
        "dest_hsm_module": "hsm_destination",
},
    "body":{
        "session_label": "session_identifier",
        "credentials":{
            "admin_pin": "admin_access_code",
            "user_pin": "user_access_code"
        },
        "key_info":{
            "type": "generated_key_type",
            "length": "generated_key_length",
            "label": "generated_key_label",
            "id": "generated_key_id"
        },
        "csr_details"{
            "csr": "csr_data",
            "csr_file": "csr_path"
        },
        "cert_details": {
             "hash_algorithm": "signing_hash_algorithm",
             "days": "certificate_validity"
         }
    }
}
```

Figure 4.1: ADCP Standard JSON Message

operations. It is typically used by the token's administrator or security officer. It allows operations such as initializing the token, creating or deleting security objects, and changing PINs. However, *"user_pin"* is associated with a specific user or application that accesses the token. It provides user-level access to the token and allows performing operations that do not require administrative privileges. It is used for authentication and to access the protected objects stored in the token, such as private keys or certificates.

Noting that the parameters of the *"key_info"* object are kept null, as these parameters would need to be populated with appropriate values according to the *"initsession"* request.

Once secure storage has been created, the HSM proceeds to generate a *Secure Storage Key* (SSK). The SSK is a cryptographic key that is stored securely within the HSM and used to encrypt the cryptographic objects that are generated and stored within the session. The length of the SSK and the encryption algorithm employed depend on the specifications of the HSM being used. For example, HSMs such as ENTRUST's nShield Connect HSM and Thales' Luna HSMs utilize a 256-bit SSK, and employ the Advanced Encryption Standard (AES) algorithm for encrypting the generated objects prior to storing them in the secure storage [47],[48]. The SSK provides an additional layer of security by ensuring that the stored cryptographic objects are protected and can only be accessed and decrypted by authorized entities with the proper key management and authentication mechanisms in place.

Aside from encrypting the cryptographic objects stored within the secure storage, HSMs provide a tamper-resistance mechanism that acts as a crucial line of defense against unauthorized physical access and tampering attempts. This mechanism ensures the integrity and confidentiality of sensitive cryptographic material and prevents unauthorized manipulation or extraction of key information. By integrating a range of internal sensors, HSMs are designed to detect and defend against attacks involving drilling, extreme temperatures, as well as low and high-voltage manipulations [49]. These sensors continuously monitor the physical environment of the HSM, allowing it to halt all its procedures, record events to the log, and wait to be restarted as a response to any suspicious activity or tampering attempts [50].

To achieve the intended functionality, the message traverses multiple layers of the architecture.

The application layer constructs the message with the required parameters (). The management layer adds additional information, such as source and destination details, to facilitate proper routing and tracking. The cryptography manager ensures the secure handling of cryptographic operations, while the HSM module executes the requested command within the specified module.

```
{
    "header": {
        "message_direction": "request",
        "message_id": "1",
        "message_type": "initsession",
        "source_mac_address": "00:11:22:33:44:55",
        "source_ip_address": "1.2.3.4",
        "timestamp": "2023-12-13T10:00:00.00",
        "dest_hsm_module": "nshieldhsm.so"
    },
    "body": {
        "session_label": "SecureSession1",
        "credentials": {
            "admin_pin": "admin123",
            "user_pin": "user456"
        },
        "key_info": null
    }
}
```

Figure 4.2: Secure Storage Initialization JSON Message

4.2.2 Key-Pair Generation:

Another provided security service is the key pair generation message. It is designed to generate a key pair within the specified token for cryptographic operations. It includes various parameters to provide necessary information for the operation. As with every message, the header of the key pair generation message shown in Figure 4.3 contains all the essential information that remains consistent. However, the body of the message contains specific parameters related to the key pair generation operation. These parameters include the *"session_label"* to identify the session in which the key pair should be generated and the *"user_pin"*, defined in the credentials

```
{
        "header":{
        "message_direction": "request",
        "message_id": "1",
        "message_type": "keypairgen",
        "source_mac_address": "00:11:22:33:44:55",
        "source_ip_address": "1.2.3.4",
        "timestamp": "2023-12-13T10:00:00.00,
        "dest_hsm_module": "nshieldhsm.so",
    },
    "body":{
        "session_label": "SecureSession1",
        "credentials": {
            "user_pin": "user456",
        },
        "key_info": {
            "type": "rsa",
            "length": "2048",
            "label": "KeyPair1",
            "id": "9999"
        }
    }
}
```

Figure 4.3: Key-Pair Generation JSON Message

object, that is required for user-level operations and token access.

The *"key_info"* object within the body encapsulates details required details about the key pair to be generated. It specifies the *"type"* of the cryptographic algorithm, such as "RSA" or "AES", and the desired "length" of the key. The "label" parameter assigns a name or label to the generated key pair, while the "id" parameter provides an identifier for reference and retrieval purposes.

By utilizing this message structure, the key pair generation operation can be accurately defined and executed within the hardware abstraction layer, facilitating the generation and storage of cryptographic key pairs for subsequent cryptographic operations.

After defining security service messages, requests can be transmitted by the HAL. The HAL leverages the information provided in the security service messages to direct requests to the appropriate HSMs based on the predefined policy sets.

4.2.3 X.509 Certificate Generation

Moving on to one of the most important security services, X.509 certificate generation requests are a fundamental requirement for numerous cryptographic operations, holding a prominent role within SSL Everywhere to ensure secure communication between clients. Similar to previous messages, the header section of the certificate generation message shown in Figure 4.4 contains consistent and essential information. However, the body of the message holds specific parameters tailored to the certificate generation process.

Within the body section, and similar to the previous message, *session_label* is also included as an identifier for the session within which the CSR to be signed. Plus, the ”user_pin” is also defined within the credentials object, to authorize user-level operations and access to the token.

Moving on to the specific parameters for the certificate generation request, the *csr_details* object encapsulates the CSR received from the client in a text form to be signed with the root CA certificate, and the *csr_file* that denotes the path in the server where the CSR is saved. As shown in the message below, the *csr_file* is saved based on the client's MAC address for better organization and identification. Moreover, the *cert_details* object holds information related to the X.509 certificate to be generated. First, *hash_algorithm* defines the hashing algorithm to be signed the certificate after encrypting it with the private key of the root CA, as mentioned earlier, *”sha256”* is the default algorithm if not specified by the user. Second, comes the *”days”* parameter that defines the certificate validity in days. The *”365”* value indicates that this certificate is valid for 365 days from the day the certificate is generated. Noting that, this parameter is only to be specified by the SSL Everywhere administrators.

4.3 Hardware Abstraction Layer Design

Transitioning from the overview of security services, the critical path leads to the Hardware Abstraction Layer. The design of the HAL plays a crucial role in ensuring seamless communi-

```
{
     "header": {
         "message_direction": "request",
         "message_id": "1",
         "message_type": "signcsr",
         "source_mac_address": "00:11:22:33:44:55",
         "source_ip_address": "1.2.3.4",
         "timestamp": "2023-12-13T10:00:00.00,
         "dest_hsm_module": "nshieldhsm.so",
     },
     "body": {
         "session_label": "secure_session",
         "credentials": {
             "user_pin": "user456"
         },
         "
         csr_details": {
             "csr": "-----BEGIN CERTIFICATE REQUEST-----
             "CSR_Data"
             -----END CERTIFICATE REQUEST-----",
             "csr_file": "/ssleverywhere/csr/client_MAC.csr.pem"
         },
         "cert_details": {
             "hash_algorithm": "sha256",
             "days": "365"
         }
    }
}
```

Figure 4.4: Certificate Generation JSON Message

cation and interoperability between the software and hardware components of the architecture. The HAL is designed to include two crucial components: the Service Manager and the Driver Manager. The Service Manager acts as an intermediary between the controller managers and the underlying hardware, while the Driver Manager handles the execution of hardware-specific commands.

In the operational workflow, the HAL proactively communicates with the Flow Manager to obtain the policy set. This step is crucial to ensure that request handling and management are based on predefined policies. The policy set provides instructions on various aspects, such as security measures, access control, error handling, and resource utilization, based on analyzing the security requirements, capabilities, and the current utilization of each HSM. Additionally, the policy set may include guidelines for access control, specifying which entities or users have permission to perform certain operations on the HSMs.

When a request is received from the controller managers, the Service Manager takes on the responsibility of translating the request into a feature request. This translation process entails applying predefined policies established by the managers to ensure the inclusion of the appropriate *"dest_hsm_module"* parameter in the message. By adhering to these policies, the Service Manager guarantees that the feature request contains the necessary information to route it to the correct HSM.

The Driver Manager plays a vital role in the HAL architecture by maintaining a registry that maps each feature request to the appropriate HSM's command and driver. This registry serves as a lookup table, allowing the Driver Manager to identify the specific hardware command and corresponding driver that can handle the requested feature. By leveraging this mapping, the Driver Manager ensures that the feature request is directed to the appropriate hardware component for execution.

4.4 Policy Set Design

The effective operation of the HAL depends on a well-structured policy set. The policy set design is a critical aspect of the SSL Everywhere architecture, as it determines the proper allocation of requests to the available HSMs based on predefined policies. The policy set is designed to consider various factors, including the capabilities, algorithms, hash functions, and utilization of each HSM, this ensures efficient resource utilization and optimal performance.

For example, the policy set shown in Figure 4.5 includes the *"hsm"* parameter which identifies the specific HSM to which the policy applies. For each HSM, the *"capabilities"* parameter specifies the capabilities of the HSM, indicating the operations or functions it can perform. For example, the policy set may specify that HSM1 is capable of key pair generation, certificate generation, and encryption, while HSM2 can only perform key pair generation and encryption. Plus, the *"algorithms"* parameter lists the supported cryptographic algorithms for the specified HSM. It includes algorithms such as RSA, ECC, AES, and RC4, which can be used for various cryptographic operations. The *"hash_funcs"* parameter defines the supported hash functions for the HSM. Hash functions like MD5, SHA1, SHA256, and SHA512 are commonly used for data integrity and verification purposes.

Finally, the *"utilization"* parameter reflects the current utilization level of the HSM. It can be categorized into different levels, such as low, medium, or high, based on factors like the workload, performance, and resource availability of the HSM, enabling efficient decision-making in request routing.

By designing a comprehensive and well-defined policy set, the SSL Everywhere architecture can effectively optimize resource allocation, ensure compatibility with cryptographic operations, and provide secure and efficient communication within intra-domain environments.

```
{
    policies: [
    {
        "hsm": "HSM1",
        "capabilities": ["initsession", "keypairgen",
        "certificate_gen", "encryption"],
        "algorithms": ["rsa","ecc"],
        "hash_funcs": ["md5","sha1","sha256"],
        "utilization":["low","med","high"],
    },
    {
        "hsm": "HSM2",
        "capabilities": ["initsession", "keypairgen", "encryption"],
        "algorithms": ["rsa","aes","rc4"],
        "hash_funcs": ["md5","sha256","sha512"],
        "utilization": ["low","med","high"],
    }
    ]
}
```

Figure 4.5: SSL Everywhere Policy Set

4.5 API Design

Going upward in the architecture, the SSL Everywhere API design process was complemented by conducting a comparative analysis of the API development tools outlined in Section 3.2.2. The primary objective of this assessment was to pinpoint and choose the optimal tool for constructing an API that aligns with the architecture's requirements. Our evaluation focused on two key aspects of API performance: the successful responses per second rate when increasing the number of simultaneous requests, and the latency associated with each individual request.

Figure 4.6 presents a performance comparison of the aforementioned tools based on the number of responses per second that each framework can handle as the number of concurrent users increases from 0 to 40. From the results, Django demonstrates the highest responses per second rate across the board. However, FastAPI preserves a consistent responses per second rate suggesting a robust performance under concurrent user load. Conversely, the Spring and Flask frameworks exhibit fluctuations and fail to maintain a consistent rate of responses per second.

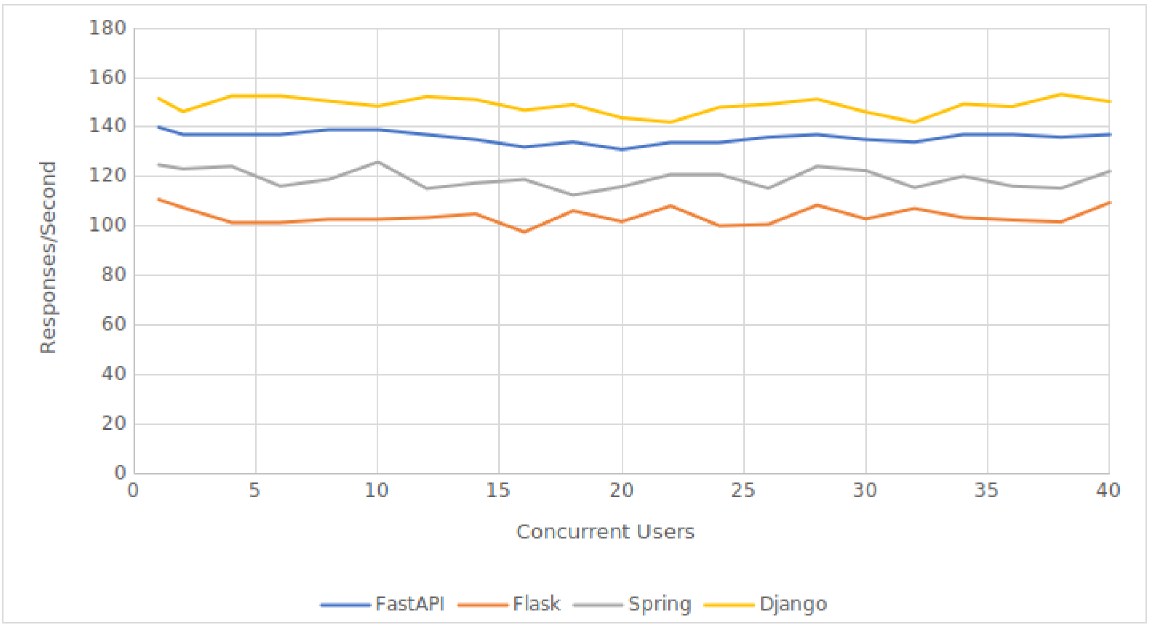


Figure 4.6: Responses/Second Comparison Results

Further, the latency comparison shown in Figure 4.7 depicts the latency, measured in milliseconds as the number of concurrent users increases from 0 to 40. As the user load rises, all frameworks exhibit an increase in latency. FastAPI shows the lowest increase, suggesting it handles additional load with minimal latency impact. Flask and Django demonstrate moderate latency growth, with Django slightly outpacing Flask at higher user counts. Spring, however, experiences the steepest rise in latency, indicating it may face challenges maintaining low latency under heavy user load. This data is crucial for understanding how each framework might affect user experience in high-traffic scenarios.

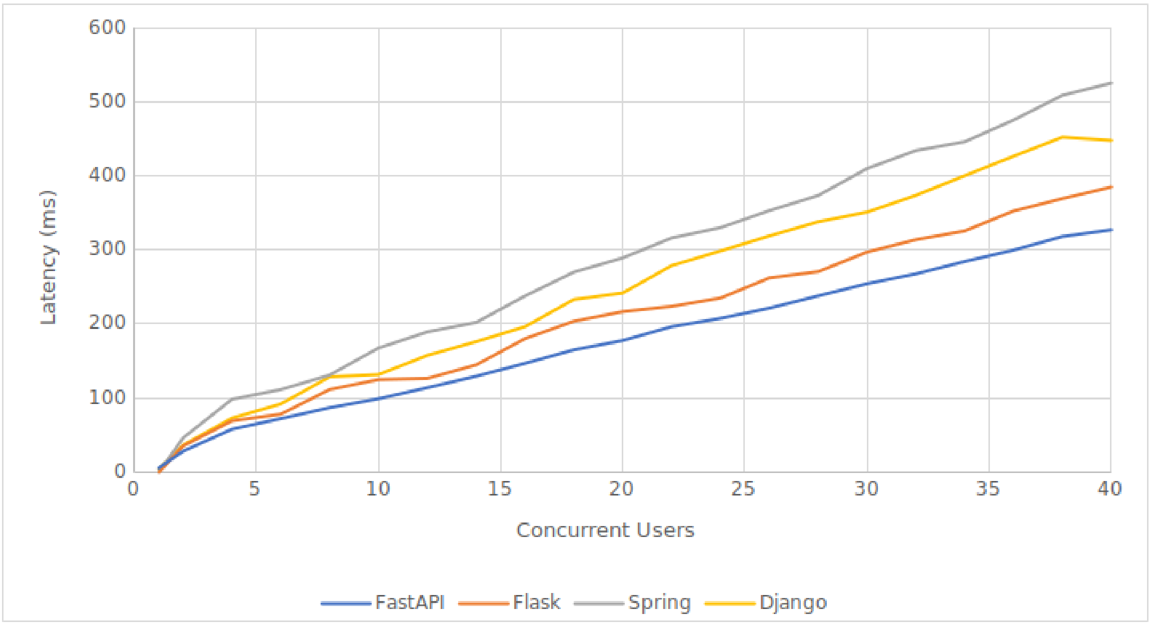


Figure 4.7: Latency Comparison Results

After evaluating the performance of various web frameworks, we have chosen FastAPI for its impressive balance of high responses per second and low latency. FastAPI's performance metrics indicate that it can handle a significant number of requests with minimal delay, which is essential for ensuring a smooth user experience. Prioritizing lower latency is crucial for our application, as it directly impacts the responsiveness and efficiency of the service.

4.6 Command-Line Tool Design

After building each of the SSL Everywhere components, the final step is to build the command-line tools that will be utilized in the software applications. The *"ssleverywhere"* command-line tool serves as a versatile interface for interacting with the SSL Everywhere architecture from the terminal. It offers various commands to initiate secure sessions, generate certificates, and perform essential cryptographic operations. The design of this tool prioritizes user-friendliness, enabling users to effortlessly harness the capabilities of the SSL Everywhere architecture.

Once installed and configured, the *"ssleverywhere"* tool generates several folders within the system, namely *"private"*, *"csr"*, and *"certs"*. Within these folders, specific functionalities are organized. The *"private"* folder serves as the repository for server key pairs. The *"csr"* folder is designated for CSRs, to be sent to the SSL Everywhere, signed, and create X.509 certificates. Finally, the *"certs"* folder stores the issued certificates obtained after successful CSR submissions.

The functionality of *"ssleverywhere"* depends on the parameters provided. The *"ssleverywhere"* tool accepts multiple parameters, each serving a specific purpose and contributing to the customization of commands. For example, the *'-gen_cert'* command is responsible for creating an X.509 certificate and saving it in the *"certs"* folder. The process starts by providing the following parameters:

- *-name*: Specifies the name associated with the certificate, this name can either be manually entered each time the *"-gen_cert"* command is executed or can be modified by administrators to remain consistent for repeated usage of the command.

- *-default*: An option that can be used with the *"-gen_cert"* command to indicate default certificate generation settings. Default parameters include key type set to *RSA*, key length set to *2048*, encryption set to *AES*, and hash set to *SHA256*.

- *-manual*: An option that can be used with the *"-gen_cert"* command to enable manual configuration of certificate generation settings.

- *-key_type*: Specifies the type of private key when manually configuring certificate generation settings.

- *-key_length*: Specifies the desired length of the private key when manually configuring certificate generation settings.

- *-encryption*: Specifies the encryption method to be used for private keys when manually configuring certificate generation settings.
- *-hash*: Specifies the hashing algorithm for cryptographic operations when manually configuring certificate generation settings.

These parameters empower users to tailor their interactions with the SSL Everywhere architecture, allowing for a flexible and efficient experience when utilizing the command-line tool.

Another command is *'-keypair_gen'*, this command is used to request SSL Everywhere to generate a cryptographic key pair and return it to the user. It accepts three parameters:

- *-label*: Specifies a label or name for the generated key pair, so the public key can be saved in the SSL Everywhere database.
- *-key_type*: Specifies the type of cryptographic key to be generated.
- *-length*: Specifies the desired length of the key.

With these parameters, users can customize the key pair generation process according to their specific requirements.

To demonstrate a usage example of the *ssleverywhere* tool, the following scenario showcases two machines on the same network utilizing *ssleverywhere* command to generate certificates, and to form a secure communication channel.

The first step is to generate two certificates for the two machines. Machine A, which acts as the server, initiates the process by running the ssleverywhere command. This command generates a certificate with the name "SE" using default settings.

```
$ ssleverywhere -gen_cert -name SE -default
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  5978  100  4965  100  1013   108k  22678 --:--:-- --:--:-- --:--:--  132k
```

Once the certificate is generated, Machine A will receive its certificate. However, Machine B, acting as the client, follows a specific configuration to generate its certificate. It runs the following command:

```
$ ssleverywhere -gen_cert -name CL -manual -key_type rsa -key_length 4096
-encryption des3 -hash sha512
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  7059  100  5331  100  1728   28048  9091 --:--:-- --:--:-- --:--:--  37951
```

This command generates a certificate with the name ”CL” using manual configuration, specifying that: the key type is RSA, the key length is set to 4096 bits, encryption is performed using the DES3 algorithm, and the hashing algorithm is SHA-512. Upon successful generation, Machine B receives its certificate.

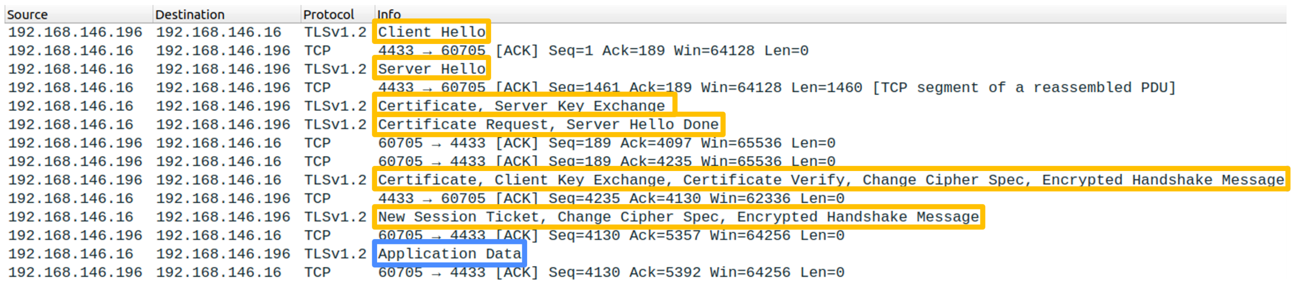
Now, both Machine A and Machine B have their respective certificates issued by the SSL Everywhere architecture, each with different configurations based on their specific needs. To put these certificates to use, the next step is to establish a secure SSL communication channel between the two machines.

Machine A takes on the role of the server in this setup. Using a socket programming Python code, it initializes the code that is designed to listen for incoming connections on a predefined port. As part of the setup process, Machine A loads its SSL certificate, private key, and root CA certificate into the SSL/TLS server context. This server program then patiently awaits incoming connection requests from clients.

Meanwhile, Machine B assumes the role of the client. It runs a client program that is responsible for initiating a connection to Machine A’s server, targeting the specific port designated for secure communication. Prior to establishing the connection, Machine B also loads its SSL certificate, private key, and root CA certificate into its SSL/TLS client context.

With both programs running and configurations in place, the SSL handshake process, as captured by Wireshark and illustrated in Figure 4.8, unfolds when Machine B connects to Machine A.

During this handshake, the messages enclosed within the yellow rectangle depict the SSL handshake process, which involves mutual authentication. After the handshake is completed, both



```
Source          Destination     Protocol Info
192.168.146.196 192.168.146.16  TLSv1.2  Client Hello
192.168.146.16  192.168.146.196 TCP      4433 → 60705 [ACK] Seq=1 Ack=189 Win=64128 Len=0
192.168.146.16  192.168.146.196 TLSv1.2  Server Hello
192.168.146.16  192.168.146.196 TCP      4433 → 60705 [ACK] Seq=1461 Ack=189 Win=64128 Len=1460 [TCP segment of a reassembled PDU]
192.168.146.16  192.168.146.196 TLSv1.2  Certificate, Server Key Exchange
192.168.146.16  192.168.146.196 TLSv1.2  Certificate Request, Server Hello Done
192.168.146.196 192.168.146.16  TCP      60705 → 4433 [ACK] Seq=189 Ack=4097 Win=65536 Len=0
192.168.146.196 192.168.146.16  TCP      60705 → 4433 [ACK] Seq=189 Ack=4235 Win=65536 Len=0
192.168.146.196 192.168.146.16  TLSv1.2  Certificate, Client Key Exchange, Certificate Verify, Change Cipher Spec, Encrypted Handshake Message
192.168.146.16  192.168.146.196 TCP      4433 → 60705 [ACK] Seq=4235 Ack=4130 Win=62336 Len=0
192.168.146.16  192.168.146.196 TLSv1.2  New Session Ticket, Change Cipher Spec, Encrypted Handshake Message
192.168.146.196 192.168.146.16  TCP      60705 → 4433 [ACK] Seq=4130 Ack=5357 Win=64256 Len=0
192.168.146.16  192.168.146.196 TLSv1.2  Application Data
192.168.146.196 192.168.146.16  TCP      60705 → 4433 [ACK] Seq=4130 Ack=5392 Win=64256 Len=0
```

Figure 4.8: Captured Handshake Process

the client and server form a secure channel and can exchange application data securely, which is encrypted and integrity-protected.

Chapter 5

Performance Evaluation

This chapter presents a performance assessment of an SSL Everywhere deployment in comparison to a conventional benchmarking tool that is typically employed for cryptographic operations. Measurements will be based on certificate generation requests and several key performance indicators. This assessment aims to provide information on the efficiency and effectiveness of SSL Everywhere in the context of certificate generation.

We begin by presenting the experimental setup, which details the controlled environment in which our tests were conducted, followed by an exploration of the evaluation metrics employed to measure the architecture's performance. Finally, we present the results of our experiments and engage in a thorough discussion to draw meaningful conclusions and implications from our findings. This analysis is intended to provide insight into the performance of SSL Everywhere by answering these questions:

- How many requests can the SSL Everywhere API handle as demand increases? (Discussed in Section 5.3.1)
- How would the average response time be affected when the demand increases? (Discussed in Section 5.3.1)
- What is the average latency for each request when increasing the demand? (Discussed

Table 5.1: Hardware Specifications

Specification	**Details**
Operating System	Ubuntu 22.04
Processor	AMD Ryzen 7
Memory	16GB
Hard Drive	1TB SSD

in Section 5.3.1)

- How much will the architecture consume from the system's and network's resources to fulfill an increasing number of requests? (Discussed in Section 5.3.2 & Section 5.3.3)

Answering those questions by generating statistics graphs for various metrics will provide a glimpse into the architecture's performance, and address its weaknesses, thus helping us identify areas that require improvement.

5.1 Experimental Setup

To start the analysis, it is imperative to understand the experimental setup used to evaluate the SSL Everywhere architecture. This section outlines the configuration and environment used to conduct the testing experiments, including the specifications of the utilized hardware to run the architecture, the software used to perform comparisons and run the tests, and the testing approaches.

5.1.1 Hardware Specifications

We start our exploration by delving into the hardware specifications used in our experiments. In order to assess the efficiency of SSL Everywhere in handling certificate generation requests, the architecture was deployed and run on a PC with the specifications mentioned in Table 5.1.

In the context of creating a controlled testing environment, it is important to note that the testing tool used to generate the certificate requests and perform the benchmarking was carried out on a separate PC connected to the same network. This separation allowed us to maintain a controlled and independent testing environment while keeping the focus on evaluating the performance of SSL Everywhere on the specified PC as outlined in Table 5.1.

5.1.2 Software Tools

Continuing from this context, to perform a valid performance evaluation of the SSL Everywhere architecture, it has been put to a comparison with *OpenSSL*, a benchmarking tool for performing cryptographic operations. *OpenSSL* is a comprehensive cryptographic software and command-line tool that is well known for its open-source implementation of the TLS protocol. Users and developers can use *OpenSSL* to perform a wide range of cryptographic functions, such as generating key pairs, CSRs, and more [51]. Noting that, the sole usage of *OpenSSL* in sensitive intra-domain environments to act as a root of trust for certificate signing and generation introduces security risks, as there have been 177 vulnerabilities reported in OpenSSL through the period of 2002 till 2019 [52].

Performance analysis for SSL Everywhere architecture and OpenSSL tool was carried out using *loadtest*. *Loadtest* [53] is a command-line tool and Node.js module used for load testing and benchmarking web applications and HTTP services. It allows the simulation of a high volume of traffic to a web application to assess its performance, identify bottlenecks, and understand how it handles various levels of load.

Loadtest operates in various modes, such as by specifying the requested *Requests-Per-Second* (RPS), the total number of requests to be processed, the total time to send as many requests as the API can handle, or the number of concurrent users. In addition, it provides detailed metrics and reporting, including RPS, mean latency, and error rates. Figure 5.1 below shows a template of a *loadtest* command along with the associated metric results.

```
\$ loadtest -n <requests> -c <concurrency> http://<API_URL>
Requests: <#_Requests>, requests per second: <RPS>, mean latency: <Latency>ms

Target URL:          http://<API_URL>
Max time (s):        <max_time_seconds>
Concurrent clients:  <concurrent_clients>
Agent:               <your_agent_name>

Completed requests:  <completed_requests>
Total errors:        <total_errors>
Total time:          <total_time_seconds>
Mean latency:        <mean_latency> ms
Effective rps:       <effective_requests_per_second>

Percentage of requests served within a certain time
  50%      <50th_percentile_latency> ms
  90%      <90th_percentile_latency> ms
  95%      <95th_percentile_latency> ms
  99%      <99th_percentile_latency> ms
 100%      <longest_request_latency> (longest request)
```

Figure 5.1: *Loadtest* Tool Output

Additional tools like the *top* (table of processes) and *nload* command-line tools were employed to monitor performance metrics beyond what loadtest offers.

The *top* command provides real-time insight into active processes within an operating system, presenting a summarized system information view that includes resource utilization data, such as CPU and memory usage [54].

The *nload* tool is specifically designed for real-time network traffic and bandwidth monitoring. It facilitates the tracking of incoming and outgoing data flows through statistical data and graphical representations, offering comprehensive insights, including the total volume of data transferred and peak network utilization [55].

5.1.3 Testing Scenarios

In the pursuit of precise experimentation, and to reduce interference, a different PC was used to carry out all test scenarios using *loadtest*. These scenarios mainly involved varying two parameters, which are:

1. Concurrent Users: It refers to the number of virtual or simulated users that are actively making requests to the target application or API at the same time. These virtual users act as if they were real users interacting with the system, generating concurrent traffic.

2. Number of Requests: It indicates the total number of HTTP requests that will be generated and sent to the API during the load testing session. This metric quantifies the volume of requests that the testing tool will simulate to assess how the system responds to various levels of traffic.

By executing these scenarios on a dedicated system, we were able to ensure that the test results accurately reflected the API's performance under different traffic conditions, from light loads to high concurrency. It is important to note that the testing scenarios involving the effective RPS are conducted under the assumption that 3 HSMs are being operated, and the load is distributed among them.

5.2 Evaluation Metrics

After discussing the experimental setup deployed to run the evaluation, it is important to highlight that the assessment of the SSL Everywhere architecture's performance relies on a set of well-defined evaluation metrics, centered around certificate generation requests. These metrics serve as the benchmark against which we measure the efficiency, reliability, and scalability of the system in the context of secure certificate generation. By systematically quantifying various aspects of performance during these certificate generation requests, we gain valuable insights into the system's behavior under different scenarios and workloads. This section outlines the key evaluation metrics used in our study and explains their significance in the context of our performance evaluation.

- Effective RPS: Effective RPS measures the rate at which the system can handle incoming certificate generation requests while considering factors like concurrent users. It is a critical metric for assessing the application's throughput and scalability in the context of secure certificate generation. Effective RPS provides valuable insights into how efficiently the system can process a specific number of certificate requests per second

- Latency: Latency refers to the time delay or the elapsed time between the initiation of a request and the moment when a response is received. It encompasses the time required for a single request to be transmitted, processed by the system, and for the corresponding response to return. It is a fundamental metric for assessing the responsiveness of the system. Lower latency is desirable as it signifies faster response times

- Processing Time: Processing time represents the time it takes for the system to handle and process all sent requests from the moment they are received until the responses are generated. It includes time spent in the application logic, database queries, and other processing steps.

- CPU Usage: Using the *top* command, CPU usage measures the amount of resources consumed by the central processing unit (CPU) during the test. High CPU usage can indicate that the system is resource-intensive and may struggle to handle additional load. Monitoring CPU usage is essential to optimize resource allocation and prevent system overload.

- Network Bandwidth: Network bandwidth measures the amount of data transmitted over the network during the load test using the *nload* tool. Monitoring network bandwidth is important for understanding the network's capacity and ensuring that it can handle the required data transfer rates.

Systematically evaluating the aforementioned metrics provides an understanding of the SSL Everywhere architecture's behavior under various scenarios and workloads, which in turn helps in identifying areas of improvement.

5.3 Results & Discussion

After establishing a foundation on the selected evaluation metrics, this section provides a comprehensive analysis of the performance evaluation conducted on the SSL Everywhere architecture. The results obtained from the experiments, as well as their implications, are presented and discussed in detail. By delving into the data and their significance, we aim to shed light on the strengths, weaknesses, and overall effectiveness of the SSL Everywhere architecture in the context of certificate generation and secure communication. This section also addresses how the architecture compares to OpenSSL and the implications for real-world applications.

5.3.1 Scalability Assessment

Starting with the results, our evaluation of scalability entails the analysis of three core metrics: effective Requests Per Second (RPS), latency, and processing time. To assess effective RPS and latency, we systematically increased the concurrent users parameter. To ensure a more comprehensive realistic assessment, each test that involved varying concurrent users ran for 10 seconds and monitored the corresponding values. For effective RPS, the value of RPS was given by getting the total number of successful requests and dividing them by 10. For the latency metric, the average latency was given for all successful requests for each test run.

From the results shown in Figure 5.2, we observed that both SSL Everywhere and OpenSSL have a roughly steady and consistent RPS rate as the number of concurrent users increased. This finding suggests that both systems demonstrated a remarkable ability to handle an increasing workload without a significant drop in RPS, emphasizing their scalability under the test conditions. However, a notable trend has become evident. SSL Everywhere consistently exhibits a higher RPS compared to OpenSSL at various user load levels. This indicates the remarkable efficiency and scalability of SSL Everywhere in processing a greater number of requests in parallel.

Throughout all testing scenarios, SSL Everywhere was able to successfully respond to a maximum of 111 requests/second. Conversely, even the minimum RPS observed for SSL Everywhere is a commendable 104 requests/second, which significantly surpasses the maximum

RPS of 83 achieved by OpenSSL. This stark contrast underscores SSL Everywhere's dominance in terms of peak RPS. The architecture consistently excels not only in its highest RPS achievements but also maintains a high average of 106 RPS. This outstanding performance further highlights the reliability, efficiency, and scalability of SSL Everywhere, making it a compelling choice for applications demanding rapid and parallelized certificate generation requests. In comparison, OpenSSL, while demonstrating its competence with an average RPS of 81, falls behind in both peak and average RPS when compared to SSL Everywhere. OpenSSL's maximum RPS, while respectable at 83, is notably outperformed by SSL Everywhere's baseline RPS, which, even at its minimum, remains higher at 104.

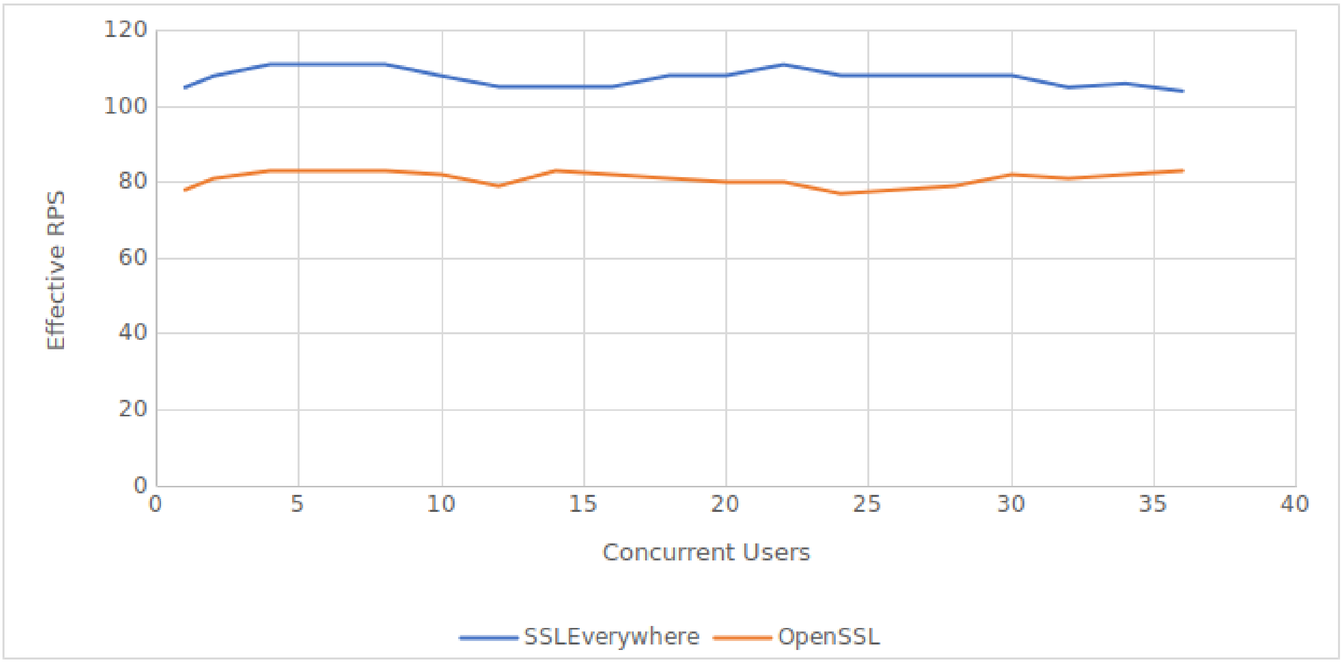


Figure 5.2: Effective RPS Results

Despite SSL Everywhere's higher RPS, it is essential to consider the corresponding latency values, as shown in Figure 5.3. Our analysis reveals that as the number of concurrent users increases, both SSL Everywhere and OpenSSL exhibit a gradual increase in latency, which is expected as system loads intensify. However, a noteworthy distinction emerges between the two. OpenSSL consistently records higher latency compared to SSL Everywhere, and this divergence becomes more pronounced as the user load intensifies.

On average, the latency for each request served by OpenSSL tends to be 13.08% higher than that of SSL Everywhere, especially as the concurrent user count rises. Despite the significant contrast in effective RPS between both architectures, the relatively slight disparity in latency can be ascribed to the inherent architectural complexity of SSL Everywhere, which inherently involves a more complex sequence of processes and database operations for each certificate generation request. Those operations include, but are not limited to, checking if the selected HSM is initialized, retrieving user's information, updating user's request count, and submitting users's certificate to be saved.

An increase in latency for each request when concurrent users increase indicates that SSL Everywhere does not employ parallelization methods to process requests simultaneously. In this context, it means that each request is handled sequentially, one after the other, as opposed to running multiple requests in parallel. The gradual rise in latency signifies that as the load intensifies, each new request may need to wait in line, leading to longer response times.

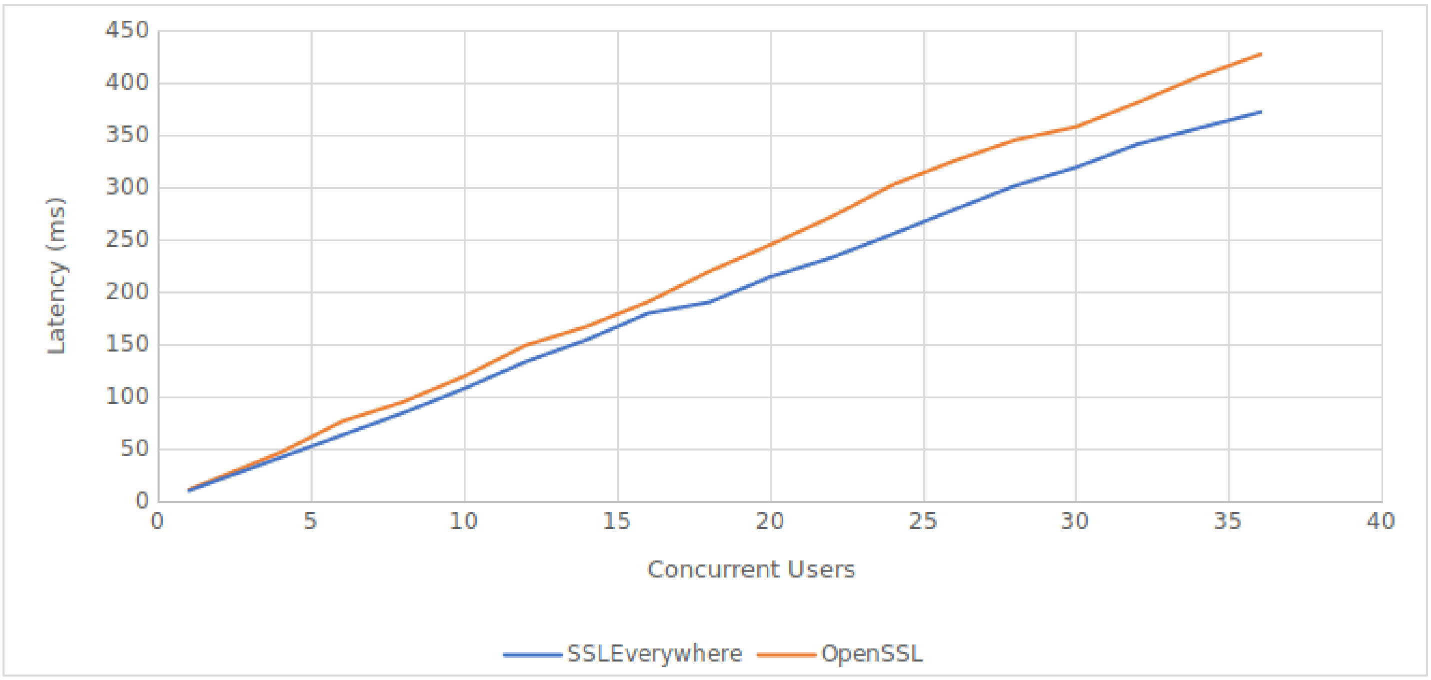


Figure 5.3: Latency Results

In addition to investigating latency, the assessment of processing time provides additional information on the performance characteristics of SSL Everywhere and OpenSSL. It's essential to note that this assessment primarily focuses on the effect of increasing the number of requests rather than concurrent users, as we aim to gauge the total processing time across different workloads. The results shown in Figure 5.4 clearly show that as the number of requests increases, both SSL Everywhere and OpenSSL experience a gradual increase in processing time, a trend to be expected as the demand for certificate generation grows.

Notably, OpenSSL consistently records a processing time that is, on average, 23.2% higher than that of SSL Everywhere as the number of requests increases. Similar to the latency test, this insignificant difference in processing time highlights the architectural and operational variances between the two systems.

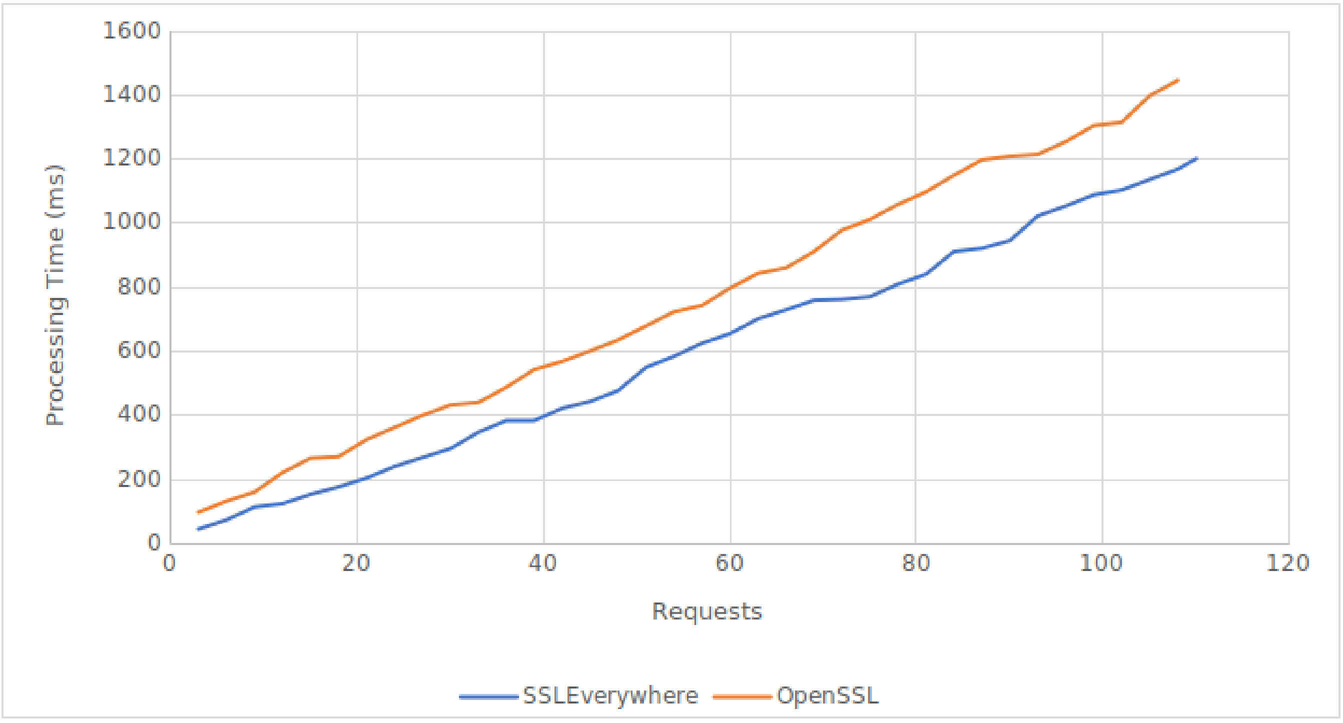


Figure 5.4: Processing Time Results

5.3.2 Resource Utilization

In addition to scalability, another evaluation criterion is the evaluation of resource utilization, with a specific focus on CPU usage, sheds light on how efficiently SSL Everywhere and OpenSSL manage system resources as the number of sent requests increases, while stressing both architectures and eventually sending 500 requests to be processed. This assessment occurs as a stress test as the volume of sent requests increases, subjecting both architectures to escalating loads, eventually processing a total of 500 requests.

CPU usage was determined using *top* command by closely monitoring essential components, namely Python, Google Chrome, and the MySQL database for the SSL Everywhere architecture. This thorough analysis offers a comprehensive perspective on the CPU demands imposed on the system during the certificate generation process.

An interesting observation from the results shown in Figure 5.5 is the clear and consistent logarithmic growth exhibited by SSL Everywhere, and a linear trend by OpenSSL. However, SSL Everywhere's growth in CPU usage is notably more pronounced compared to OpenSSL since SSL Everywhere includes more complex backend processes, such as generating the ADCP message, forwarding the message to multiple managers, generating the policy set, and accessing databases to save or retrieve information.

Throughout the tests, SSL Everywhere showcases an ability to efficiently utilize CPU resources, as evidenced by the substantial growth in CPU usage. This growth, although higher, eventually stabilizes at a maximum of 15.1%. This is because the application or processes included in SSL Everywhere may be configured with resource allocation limits that restrict its ability to consume more than a certain percentage of the CPU's total capacity.

5.3.3 Network Usage

Another evaluation criterion, which involves the use of the *nload* tool, offers crucial insights into network utilization. This evaluation concentrates on understanding the influence of certificate generation on network resources, specifically focusing on SSL Everywhere. Notably, OpenSSL operates and is utilized locally within each server, rendering the monitoring of network utilization primarily relevant to SSL Everywhere.

Throughout our experiments shown in Figure 5.6, we observed a consistent upward trajectory in network bandwidth consumption by SSL Everywhere. As the number of concurrent users increased, so did the utilization of the network bandwidth. This growth in network utilization is

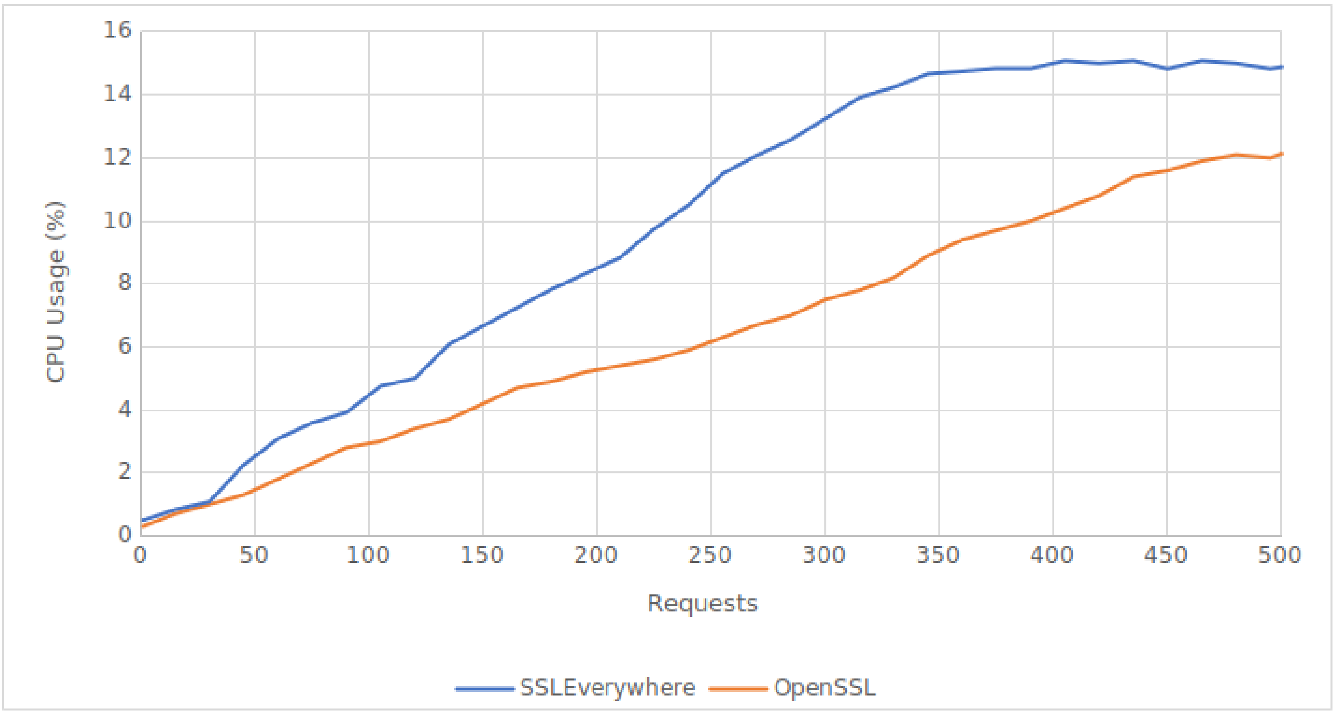


Figure 5.5: CPU Usage Results

in line with the increased demand for certificate generation requests. However, it is noteworthy that the network bandwidth utilization eventually reaches a ceiling, stabilizing at approximately 690-700 KB/s. This observed cap indicates that SSL Everywhere, while efficiently scaling to meet growing demands, has an upper limit in terms of network resource utilization. The stability in network bandwidth at this level underscores the architecture's ability to effectively allocate and manage network resources without excessive saturation.

Consistency in network bandwidth, as seen in Figure 5.6 offers a great advantage, as it leads to efficient resource management and improved quality of service, particularly for applications requiring a consistent level of bandwidth. Furthermore, maintaining consistent network bandwidth is crucial to enhancing user experience by providing reliable and uninterrupted access to the security services provided by SSL Everywhere. In addition, it helps in security efforts, as unexpected spikes in bandwidth usage can signal potential threats [56].

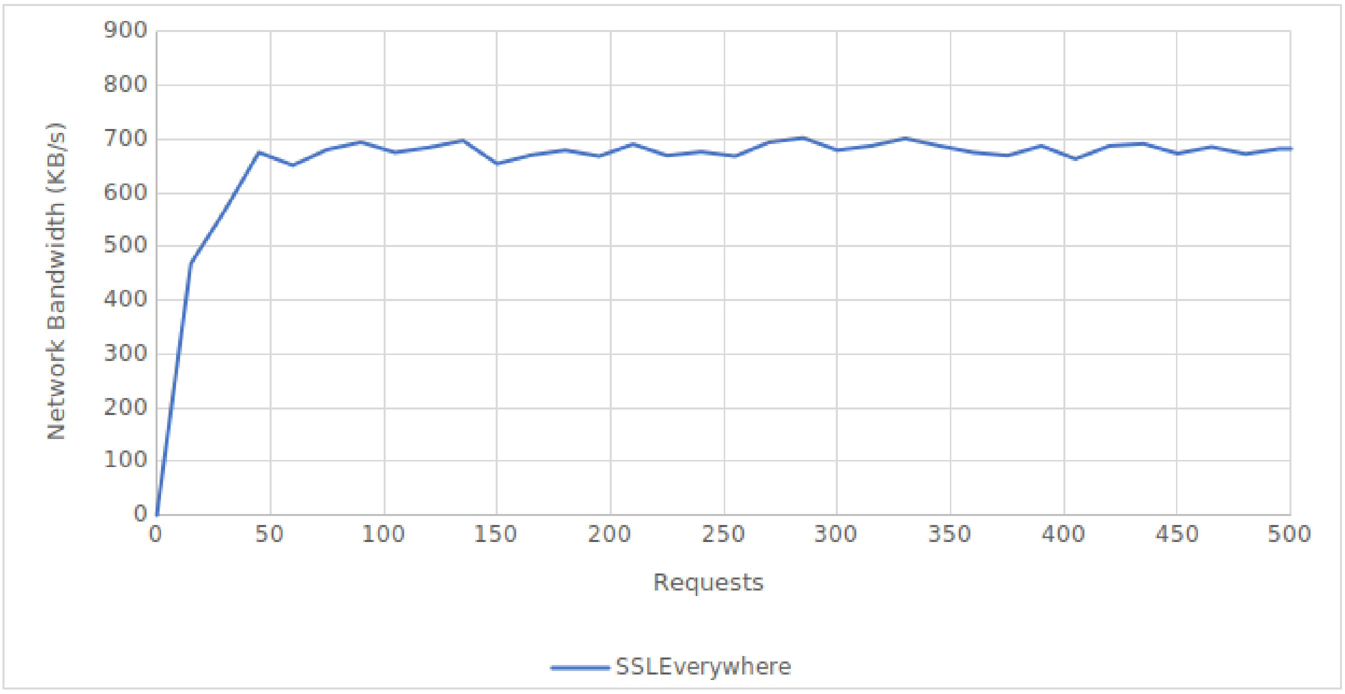


Figure 5.6: Network Bandwidth Results

5.3.4 Discussion

Following the evaluation of the aforementioned criteria, we proceed to discuss our findings. The performance evaluation of SSL Everywhere and its comparison with OpenSSL have revealed valuable insights into the strengths and areas for improvement of the SSL Everywhere architecture. This discussion section delves into the findings and outlines strategies to enhance SSL Everywhere's performance, addressing the observed weaknesses.

Parallelization

The first notable weakness observed in SSL Everywhere is the increase in latency and processing time for each test as the number of concurrent users and requests increases, as shown in Figure 5.3 and Figure 5.4. This suggests an opportunity for improvement through the implementation of parallelization and multithreading methods. By introducing parallel processing, SSL Everywhere can reduce latency and processing time by enabling the simultaneous execution of multiple certificate generation requests. This approach can enhance the system's responsiveness under heavy workloads and significantly improve the user experience, particularly in real-time or interactive applications.

Database Caching

Another area of concern revolves around SSL Everywhere's CPU usage growth, which, eventually stabilizes at a reasonable level, presents room to be further optimized. To reduce CUP utilization caused by MySQL processes, one strategy is to implement database caching mechanisms for MySQL database processes. Caching frequently accessed data can reduce the computational load on the CPU by minimizing the need to repeatedly retrieve data from the database. This optimization can lead to more efficient resource utilization and ultimately lower CPU usage.

Request Throttling

Another area of improvement is to perform request throttling. To manage system resource demands and maintain a consistent level of performance, it may be beneficial to implement controls on the number of requests sent by each user in a period of time. By limiting the number of requests from a single user, SSL Everywhere can prevent potential spikes in resource usage and maintain a smoother and more predictable operation. This approach can help ensure fair resource allocation, reduce the risk of resource contention, and mitigate the impact of excessive concurrent requests on system performance.

In conclusion, the weaknesses observed in SSL Everywhere present opportunities for improvement and refinement. Implementing parallelization methods, optimizing CPU usage through caching, and introducing request limits per user are viable strategies to address these weaknesses and further elevate SSL Everywhere's performance and efficiency in providing various security services. These improvements can position SSL Everywhere as a robust solution for a wide range of demanding real-world applications.

Chapter 6

Conclusion & Future Work

6.1 Conclusion

In an era marked by ever-increasing digitalization and networked communication, security and privacy are of paramount concern. The SSL Everywhere architecture has appeared as a novel solution to address the evolving challenges of secure communication within intra-domain environments. This research has discussed the architecture's design, components, and performance evaluation, providing valuable insights into its capabilities and contributions.

The primary objective of this research was to study and examine the security challenges within intra-domain environments, along with industrial and academic solutions. This led to the development of a centralized architecture that improves security within intra-domain environments. In the course of this research, the following objectives have been successfully achieved:

1. Investigate current security challenges in intra-domain communication.
2. Study and review networking protocols deployed in intra-domain environments along with their security mechanisms.
3. Examine the most recent academic and industrial solutions that address the weaknesses of intra-domain communication.

4. Identify the weaknesses and limitations within the security measures of individual networking protocols, as well as the shortcomings in existing solutions.

5. Propose and design the SSL Everywhere architecture that overcomes the aforementioned weaknesses and limitations.

6. Develop and build the SSL Everywhere architecture through a bottom-up approach.

7. Evaluate the SSL Everywhere's performance through a comparative analysis with a traditional solution, considering factors such as latency, throughput, and resource utilization.

8. Point out weaknesses in the SSL Everywhere and identify areas for improvement.

Moreover, SSL Everywhere's integration of Hardware Security Modules has contributed to its robustness by providing secure storage and key management. This implementation ensures that cryptographic keys and sensitive data are protected against unauthorized access and tampering, reinforcing the architecture's commitment to security.

Further, through detailed design and careful consideration of secure communication principles, SSL Everywhere has demonstrated its ability to provide encryption, authentication, and integrity, thus safeguarding sensitive data during transmission. Table 6.1 illustrates how SSL Everywhere successfully fills the security gaps across the discussed intra-domain communication protocols, providing an all-encompassing mechanism to cover current security weaknesses. Plus, after running a series of performance assessments, the architecture's API design has proven to be efficient and adaptable to provide a seamless user experience for various security services.

In summary, this research serves as a foundational step toward achieving the goal of secure communication within intra-domain environments. It is expected that ongoing developments and refinements will continue to strengthen SSL Everywhere's position as a valuable tool in the realm of secure networking.

Table 6.1: Comparison Between Security Mechanisms of Mentioned Networking Protocols

Protocol	Security Mechanism	Peer-to-Peer	Achieves Encryption	SSL-Based	Forces Encryption
HTTP	HTTPS	Yes	Yes	Yes	No
	HTTPS/HSTS	No	Yes	Yes	Yes
SMTP	STARTTLS	Yes	Yes	Yes	No
TCP	TCPCrypt	Yes	Yes	No	No
DHCP	DHCP Snooping	No	No	-	-
SSL Everywhere	-	Yes	Yes	Yes	Yes

6.2 Future Work

While SSL Everywhere represents a significant advancement in addressing the security challenges within intra-domain environments, several avenues for future research and development can further enhance its capabilities and impact. Some potential areas for future work include:

- **Integration and Compatibility with Diverse HSMs**: Future work can focus on enhancing SSL Everywhere's compatibility with a broader range of HSMs, allowing organizations to choose from a diverse set of HSM options while maintaining the architecture's security standards and principles.

- **Enhanced Performance Optimization**: As mentioned in 5.3.4, continued efforts can be made to optimize SSL Everywhere's performance, particularly in scenarios with a high volume of concurrent connections. Further research may explore advanced load balancing techniques, caching mechanisms, and parallel processing to reduce latency and improve throughput.

- **Enhanced Security Analytics**: Improving the architecture's capabilities for monitoring and analyzing security events in real-time can be beneficial. Future work can explore the integration of advanced security information and event management (SIEM) [57] systems for proactive threat detection and response.

- **Standardization and Industry Adoption**: Future work can involve efforts to promote SSL Everywhere as an industry-standard solution for secure intra-domain communication. Collaboration with relevant standardization bodies and industry organizations may be essential to achieving widespread adoption.

This thesis is a component of a broader research initiative aimed at introducing innovative solutions to enhance the security and efficiency of communication within intra-domain environments. As part of this ongoing effort, several areas of potential future work have been identified to further advance the SSL Everywhere architecture and address emerging challenges.

Bibliography

[1] M. Zhang, "On the State of the Inter-domain and Intra-domain Routing Security," en, p. 23,

[2] H. Schulze, *2020 insider threat report*. [Online]. Available: https://www.cybersecurity-insiders.com/wp-content/uploads/2019/11/2020-Insider-Threat-Report-Gurucul.pdf.

[3] T. Bourke, *Server load balancing*, en, 1st ed. Beijing ; Sebastopol, Calif: O'Reilly, 2001, OCLC: ocm47890317, ISBN: 978-0-596-00050-9.

[4] P. Membrey, D. Hows, and E. Plugge, "SSL Load Balancing," in *Practical Load Balancing*, Berkeley, CA: Apress. DOI: 10.1007/978-1-4302-3681-8_11. [Online]. Available: http://link.springer.com/10.1007/978-1-4302-3681-8_11.

[5] *SSL certificate for IP address - an expert guide on SSL for IP address*, Sep. 2022. [Online]. Available: https://sectigostore.com/page/ssl-certificate-for-ip-address/.

[6] *Using an IP address in an SSL certificate*. [Online]. Available: https://www.geocerts.com/support/ip-address-in-ssl-certificate.

[7] A. Herzberg, M. Hollick, and A. Perrig, "Secure Routing for Future Communication Networks (Dagstuhl Seminar 15102)," DOI: 10.4230/DAGREP.5.3.28. [Online]. Available: http://drops.dagstuhl.de/opus/volltexte/2015/5267/.

[8] C. A. Shue, A. J. Kalafut, and M. Gupta, "A Unified Approach to Intra-domain Security," in *2009 International Conference on Computational Science and Engineering*, Vancouver, BC, Canada, 2009. DOI: 10.1109/CSE.2009.204. [Online]. Available: http://ieeexplore.ieee.org/document/5283540/.

[9] P. D. Boisrond, *To Terminate or Not to Terminate Secure Sockets Layer (SSL) Traffic at the Load Balancer*, Nov. 2020.

[10] Y. Aref and A. Ouda, “Autonomous vehicle cyber-attacks classification framework,” in *2023 15th International Conference on COMmunication Systems & NETworkS (COMSNETS)*, 2023, pp. 373–377. DOI: 10.1109/COMSNETS56262.2023.10041387.

[11] *What is a secure communication?* [Online]. Available: https://docs.huihoo.com/globus/gt3-tutorial/ch10s01.html#:~:text=Most%20authors%20consider%20the%20three,privacy%2C%20integrity%2C%20and%20authentication..

[12] S. And Communication Networks, “Retracted: Application Research of Data Encryption Technology in Computer Network Information Security,” en, *Security and Communication Networks*, vol. 2023, pp. 1–1, Jul. 2023, ISSN: 1939-0122, 1939-0114. DOI: 10.1155/2023/9873584. [Online]. Available: https://www.hindawi.com/journals/scn/2023/9873584/ (visited on 11/13/2023).

[13] Xuguang Ren and Xin-Wen Wu, “A novel dynamic user authentication scheme,” en, in *2012 International Symposium on Communications and Information Technologies (ISCIT)*, Gold Coast, Australia: IEEE, Oct. 2012, pp. 713–717, ISBN: 978-1-4673-1157-1 978-1-4673-1156-4 978-1-4673-1155-7. DOI: 10.1109/ISCIT.2012.6380995. [Online]. Available: http://ieeexplore.ieee.org/document/6380995/ (visited on 11/13/2023).

[14] L. Yin and Y. Guo, “Research on Quantitative Evaluation for Integrity,” en, in *2009 Fifth International Conference on Information Assurance and Security*, Xi’An China: IEEE, 2009, pp. 689–692, ISBN: 978-0-7695-3744-3. DOI: 10.1109/IAS.2009.105. [Online]. Available: http://ieeexplore.ieee.org/document/5284215/ (visited on 11/13/2023).

[15] R. Dastres and M. Soori, “Secure socket layer (ssl) in the network and web security,” *International Journal of Computer and Information Sciences*, vol. 14, pp. 330–333, Oct. 2020.

[16] A. S. Gillis and S. Shea, *What is an x.509 certificate?* Jun. 2022. [Online]. Available: https://www.techtarget.com/searchsecurity/definition/X509-certificate.

[17] L. E. Hughes, “Pkcs #10 certificate-signing request (csr),” in *Pro Active Directory Certificate Services: Creating and Managing Digital Certificates for Use in Microsoft Networks*. Berkeley, CA: Apress, 2022, pp. 75–91, ISBN: 978-1-4842-7486-6. DOI: 10.1007/978-1-4842-7486-6_6. [Online]. Available: https://doi.org/10.1007/978-1-4842-7486-6_6.

[18] J. Han, S. Kim, T. Kim, and D. Han, "Toward scaling hardware security module for emerging cloud services," in *Proceedings of the 4th Workshop on System Software for Trusted Execution*, 2019. DOI: 10.1145/3342559.3365335. [Online]. Available: https://doi.org/10.1145/3342559.3365335.

[19] N. I. of Standards and Technology, "Security requirements for cryptographic modules,"

[20] MIT, *MIT Notes; Application Layer*. [Online]. Available: https://www.cs.uct.ac.za/mit_notes/networks/htmls/chp02.html.

[21] E. Rescorla, *HTTP Over TLS*, RFC 2818 (Informational), RFC, Obsoleted by RFC 9110, updated by RFCs 5785, 7230, Fremont, CA, USA: RFC Editor, May 2000. DOI: 10.17487/RFC2818. [Online]. Available: https://www.rfc-editor.org/rfc/rfc2818.txt.

[22] *Usage statistics of Default protocol https for websites*. [Online]. Available: https://w3techs.com/technologies/details/ce-httpsdefault.

[23] M. Koop, E. Tews, and S. Katzenbeisser, "In-Depth Evaluation of Redirect Tracking and Link Usage," *Proceedings on Privacy Enhancing Technologies*, 2020. DOI: 10.2478/popets-2020-0079. [Online]. Available: https://petsymposium.org/popets/2020/popets-2020-0079.php.

[24] L. Chang, H.-C. Hsiao, W. Jeng, T. H.-J. Kim, and W.-H. Lin, "Security Implications of Redirection Trail in Popular Websites Worldwide," Perth Australia: International World Wide Web Conferences Steering Committee, 2017. DOI: 10.1145/3038912.3052698. [Online]. Available: https://dl.acm.org/doi/10.1145/3038912.3052698.

[25] A. R. Chordiya, S. Majumder, and A. Y. Javaid, "Man-in-the-Middle (MITM) Attack Based Hijacking of HTTP Traffic Using Open Source Tools," in *2018 IEEE International Conference on Electro/Information Technology (EIT)*, Rochester, MI: IEEE, 2018. DOI: 10.1109/EIT.2018.8500144. [Online]. Available: https://ieeexplore.ieee.org/document/8500144/.

[26] J. Hodges, C. Jackson, and A. Barth, *HTTP Strict Transport Security (HSTS)*, RFC 6797 (Proposed Standard), RFC, Fremont, CA, USA: RFC Editor, Nov. 2012. DOI: 10.17487/RFC6797. [Online]. Available: https://www.rfc-editor.org/rfc/rfc6797.txt.

[27] A. P. H. Fung and K. W. Cheung, "SSLock: Sustaining the trust on entities brought by SSL," in *Proceedings of the 5th ACM Symposium on Information, Computer and*

Communications Security - ASIACCS '10, Beijing, China: ACM Press, 2010. DOI: 10.1145/1755688.1755714. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1755688.1755714.

[28] F. Mat Nor, K. Abd Jalil, A. Abd Kadir, and J.-l. Ab Manan, "Using trusted platform module to mitigate SSL stripping," en, in *2013 IEEE Conference on Open Systems (ICOS)*, Kuching, Malaysia: IEEE, Dec. 2013, pp. 237–241, ISBN: 978-1-4799-0285-9 978-1-4799-3152-1. DOI: 10.1109/ICOS.2013.6735081. [Online]. Available: http://ieeexplore.ieee.org/document/6735081/.

[29] I. Dolnak and J. Litvik, "Introduction to HTTP security headers and implementation of HTTP strict transport security (HSTS) header for HTTPS enforcing," in *2017 15th International Conference on Emerging eLearning Technologies and Applications (ICETA)*, Stary Smokovec, 2017. DOI: 10.1109/ICETA.2017.8102478. [Online]. Available: http://ieeexplore.ieee.org/document/8102478/.

[30] J. Klensin (Ed.), *Simple Mail Transfer Protocol*, RFC 2821 (Proposed Standard), RFC, RFC Editor, 2001. DOI: 10.17487/RFC2821. [Online]. Available: https://www.rfc-editor.org/rfc/rfc2821.txt.

[31] D. Poddebniak, F. Ising, H. Böck, and S. Schinzel, "Why TLS is better without STARTTLS: A Security Analysis of STARTTLS in the Email Context,"

[32] Christopher Allen, *NEW DRAFT: Regularizing Port Numbers for SSL.* Feb. 1997. [Online]. Available: https://lists.w3.org/Archives/Public/ietf-tls/1997JanMar/0079.html.

[33] C.-l. Chan, R. Fontugne, K. Cho, and S. Goto, "Monitoring TLS adoption using backbone and edge traffic," in *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, Honolulu, HI, 2018. DOI: 10.1109/INFCOMW.2018.8406957. [Online]. Available: https://ieeexplore.ieee.org/document/8406957/.

[34] K. Moore and C. Newman, *Cleartext Considered Obsolete: Use of Transport Layer Security (TLS) for Email Submission and Access*, RFC 8314 (Proposed Standard), RFC, Updated by RFC 8997, Fremont, CA, USA: RFC Editor, Jan. 2018. DOI: 10.17487/RFC8314. [Online]. Available: https://www.rfc-editor.org/rfc/rfc8314.txt.

[35] A. Bittau, M. Hamburg, M. Handley, and D. Boneh, "The case for ubiquitous transport-level encryption,"

[36] S. A. Nikolidakis, V. Giotsas, E. Georgakakis, and D. D. Vergados, "Towards utilizing tcpcrypt in mobile healthcare applications," in *Wireless Mobile Communication and Healthcare*, Springer Berlin Heidelberg, 2012.

[37] B. Bhushan, G. Sahoo, and A. K. Rai, "Man-in-the-middle attack in wireless and computer networking — A review," in *2017 3rd International Conference on Advances in Computing,Communication & Automation (ICACCA) (Fall)*, Dehradun, 2017. DOI: 10.1109/ICACCAF.2017.8344724. [Online]. Available: https://ieeexplore.ieee.org/document/8344724/.

[38] M. Aldaoud, D. Al-Abri, A. Al Maashri, and F. Kausar, "DHCP attacking tools: An analysis," *Journal of Computer Virology and Hacking Techniques*, 2021. DOI: 10.1007/s11416-020-00374-8. [Online]. Available: https://link.springer.com/10.1007/s11416-020-00374-8.

[39] W. Odom, *Ccna 200-301 official cert guide, volume 2*, en, 1st ed. Hoboken: Pearson Education, Inc, 2019, ISBN: 978-1-58714-713-5.

[40] B. Kang and H. Choo, "An SDN-enhanced load-balancing technique in the cloud system," *The Journal of Supercomputing*, 2018. DOI: 10.1007/s11227-016-1936-z. [Online]. Available: http://link.springer.com/10.1007/s11227-016-1936-z.

[41] Z. Shu and Y. Kadobayashi, "Troubleshooting on intra-domain routing instability," 2004. DOI: 10.1145/1016687.1016699. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1016687.1016699.

[42] SerdarSoysal, *Authentication overview for sharepoint server - sharepoint server*. [Online]. Available: https://learn.microsoft.com/en-us/sharepoint/security-for-sharepoint-server/authentication-overview.

[43] W. Hupp, A. Hasandka, R. S. de Carvalho, and D. Saleem, "Module-OT: A Hardware Security Module for Operational Technology," in *2020 IEEE Texas Power and Energy Conference (TPEC)*, IEEE. DOI: 10.1109/TPEC48276.2020.9042540. [Online]. Available: https://ieeexplore.ieee.org/document/9042540/.

[44] A. Vereecke, *Koninklijke militaire school: Kms*. [Online]. Available: https://aws.amazon.com/kms/.

[45] *Cve-2020-8897 detail*. [Online]. Available: https://nvd.nist.gov/vuln/detail/CVE-2020-8897.

[46] *OpenDNSSEC SoftHSM*. [Online]. Available: https://www.opendnssec.org/softhsm/.

[47] "nShield® Connect: User Guide for Linux," Nov. 17, 2021. [Online]. Available: https://nshielddocs.entrust.com/docs/connect-ug/12.80/User_Guide_nShield_Connect_12.80_Linux.pdf.

[48] *Scalable Key Storage*. [Online]. Available: https://thalesdocs.com/gphsm/luna/7/docs/usb/Content/admin_usb/partition/SKS.htm#SMKtypes.

[49] *SafeNet HSM Tamper Detection*. [Online]. Available: https://thalesdocs.com/gphsm/luna/6.3/docs/network/Content/administration/tamper/tamper_events.htm.

[50] K. Markantonakis and K. Mayes, Eds., *Secure Smart Embedded Devices, Platforms and Applications*, New York, NY: Springer New York, 2014, ISBN: 978-1-4614-7914-7 978-1-4614-7915-4. DOI: 10.1007/978-1-4614-7915-4. [Online]. Available: https://link.springer.com/10.1007/978-1-4614-7915-4.

[51] C. Tomita, M. Takita, K. Fukushima, Y. Nakano, Y. Shiraishi, and M. Morii, "Extracting the Secrets of OpenSSL with RAMBleed," en, *Sensors*, vol. 22, no. 9, p. 3586, May 2022, ISSN: 1424-8220. DOI: 10.3390/s22093586. [Online]. Available: https://www.mdpi.com/1424-8220/22/9/3586 (visited on 11/02/2023).

[52] J. Walden, "The Impact of a Major Security Event on an Open Source Project: The Case of OpenSSL," en, in *Proceedings of the 17th International Conference on Mining Software Repositories*, arXiv:2005.14242 [cs], Jun. 2020, pp. 409–419. DOI: 10.1145/3379597.3387465. [Online]. Available: http://arxiv.org/abs/2005.14242 (visited on 11/02/2023).

[53] *"loadtest" linux tool*. [Online]. Available: https://www.npmjs.com/package/loadtest.

[54] *"top" linux command*, May 2022. [Online]. Available: https://www.geeksforgeeks.org/top-command-in-linux-with-examples/.

[55] Rolandriegel, *Nload: Real-time network traffic monitor*. [Online]. Available: https://github.com/rolandriegel/nload.

[56] G. Goos, J. Hartmanis, J. van Leeuwen, *et al.*, “Lecture Notes in Computer Science,” en,

[57] Gartner$_{I}$*nc*, *Definition of security information and event management (siem).* [Online]. Available: https://www.gartner.com/en/information-technology/glossary/security-information-and-event-management-siem#:~:text=Security%20information%20and%20event%20management%20(SIEM)%20technology%20supports%20threat%20detection,event%20and%20contextual%20data%20sources..

Curriculum Vitae

Name:	Yazan Aref
Post-Secondary Education and Degrees:	2022 - Present, M.E.Sc Electrical and Computer Engineering University of Western Ontario London, ON, Canada
	2017 - 2021, B.Eng Electrical and Electronics Engineering University of Sharjah Sharjah, United Arab Emirates
Honours and Awards:	Graduate Fellowship, 2022 - 2024 Western Graduate Research Scholarship, 2022 - 2024
Related Work Experience:	Teaching and Research Assistant The University of Western Ontario 2022 - Present
	Teaching Assistant American University of Sharjah 2021 - 2022

Publications:

[1]: Y. Aref and A. Ouda, "Autonomous Vehicle Cyber-Attacks Classification Framework", 2023 15th International Conference on COMmunication Systems and NETworkS (COMSNETS), Bangalore, India, 2023, pp. 373-377, doi: 10.1109/COMSNETS56262.2023.10041387.

[2]: Y. Aref and A. Ouda, "Still Computers Networking is Less Secure Than it Should Be,

Causes and Solution", 2023 International Symposium on Networks, Computers and Communications (ISNCC): Trust, Security and Privacy, Doha, Qatar, 2023.

[3]: Y.Aref and A. Ouda, "SSL Everywhere: Leveraging HSMs for Enhanced Intra-Domain Security", Journal Article, (In Progress)