future internet

MDPI

Article

HSM4SSL: Leveraging HSMs for Enhanced Intra-Domain Security †

Yazan Aref and Abdelkader Ouda *

Department of Electrical and Computer Engineering, Western University, London, ON N6A 5B9, Canada; yaref@uwo.ca

* Correspondence: aouda@uwo.ca

† This paper is an extended version of our paper published in the International Symposium on Networks, Computers and Communications (ISNCC), Still Computers Networking is Less Secure Than It should be, Causes and Solution, Doha, Qatar, 23–26 October 2023.

Abstract: In a world where digitization 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 paper introduces HSM4SSL, a comprehensive solution designed to address the evolving challenges of secure data transmission in intra-domain environments. By leveraging hardware security modules (HSMs), HSM4SSL 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 in the 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, HSM4SSL bridges this gap by offering a unified and comprehensive solution that includes certificate management, key management practices, and various security services. HSM4SSL comprises three layers to provide a standardized interaction between software applications and HSMs. A performance evaluation was conducted comparing HSM4SSL with a benchmark tool for cryptographic operations. The results indicate that HSM4SSL achieved 33% higher requests per second (RPS) compared to OpenSSL, along with a 13% lower latency rate. Additionally, HSM4SSL efficiently utilizes CPU and network resources, outperforming OpenSSL in various aspects. These findings highlight the effectiveness and reliability of HSM4SSL in providing secure communication within intra-domain environments, thus addressing the pressing need for enhanced security mechanisms.

Keywords: intra-domain; secure socket layer; transport layer security; public-key certificates; hardware security modules

Citation: Aref, Y.; Ouda, A. HSM4SSL: Leveraging HSMs for Enhanced Intra-Domain Security. *Future Internet* **2024**, *16*, 148. https://doi.org/10.3390/fi16050148

Academic Editors: Christoph Stach, Clémentine Gritti and Iouliana Litou

Received: 23 March 2024
Revised: 20 April 2024
Accepted: 23 April 2024
Published: 26 April 2024

1. Introduction

In an era defined by rapid technological advancement and increasing digitization, 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].

Expanding the network capacity within a domain to accommodate numerous devices with networking capabilities underscores the importance of enhancing security measures in these settings. Notably, in 2020, insider employees, frequently using compromised devices, were responsible for 68% of organizations' security incidents and malicious activities [2]. This highlights the critical need to focus on security issues not just in the public internet but also within internal networks. Moreover, organizations that manage a multitude of servers to manage multiple requests commonly deploy *load balancers* or *reverse proxies* to intercept traffic originating from multiple hosts or endpoints. These tools effectively sort out incoming requests and distribute them among the back-end servers. The significance of the load balancing process extends across all cloud environments, offering reliability and availability through continuous monitoring of server statuses and directing requests exclusively to servers that are available [3]. Furthermore, it alleviates the workload on back-end servers by avoiding the need for decrypting traffic, achieved through the utilization of *Secure Socket Layer (SSL) termination*. SSL termination not only assists in intrusion detection by identifying anomaly packets but also ensures that these packets are detected before reaching the servers [4]. Despite these advantages, it is important to note that this approach introduces security vulnerabilities by establishing insecure channels between the reverse proxy and the back-end servers, exposing communication to various potential attacks. 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 [5]. Plus, following the CA/Browser Forum meeting in March 2023, Google declared its intention to limit SSL certificate validity to 90 days instead of 398 days. Managing SSL certificates with a 90-day validity period not only adds complications to the management of certificates, particularly in scenarios where organizations operate many servers, but it also introduces increased costs associated with frequent renewals and administrative overhead [6].

With these aforementioned issues, along with our previous work's observations [7], both academia and implemented systems are currently inadequate in representing the progress in secure intra-domain communication. The reasons for this underrepresentation remain obscure. Consequently, this situation is consistently leading to numerous security breaches and attacks within organizations. The majority of intra-domain attacks stem from the prevailing lack of encryption and authentication in intra-domain communication, especially between servers [8,9]. However, certificate usage for SSL establishment between the load balancer and the servers and also between all endpoints was suggested by *Boisrond* in [10] 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.

This paper is focused on proposing a standard, comprehensive framework aimed at enhancing security within intra-domain environments. In response to the challenges posed by the inadequacies of existing security mechanisms, our framework aims to provide a holistic solution that covers all aspects of secure communication within organizational networks. The remaining sections of this paper are organized as follows: Section 2 will

present a summary of the current security mechanisms and solutions deployed within intra-domain networks to highlight the need for our work, while Section 3 will discuss our findings and present the proposed HSM4SSL architecture along with its layers, components, and provided services. In Section 4, we will examine the performance evaluation of HSM4SSL and discuss our findings, and finally, the concluding remarks will be discussed in Section 5.

2. Current Security Mechanisms and Implementations

In the realm of securing communication within and across ASs, considerable efforts have been invested in strengthening the security of individual networking protocols. The concept of SSL for intra-domain networks is well documented, with initial implementations dating back to the 1990s. These early adaptations provided the basis for secure internal communications, yet the literature has largely overlooked the need for a comprehensive and holistic approach [9]. What follows is a brief exploration of various security mechanisms and solutions designed for specific networking protocols and environments. This exploration involves a comparative analysis, shedding light on the weaknesses of intra-domain communication.

- HTTP : Initially, HTTP did not prioritize encryption, which became problematic as sensitive data increased. This led to the creation of HTTPS, which uses TLS/SSL to encrypt and authenticate exchanged communication, making most domains and servers prefer HTTPS connections [11]. In 2022, over 79% of websites defaulted to HTTPS [12]. Web browsers enforce HTTPS usage through two main methods. The former is HTTPS redirection, which is utilized when a website undergoes updates or transitions to a new URL or when a server forcefully redirects communication from HTTP to HTTPS. All browsers inform users about secure channel establishment issues, allowing users to decide whether to continue interacting with the server despite potential security concerns [13]. However, depending solely on user decisions, using "Click Through" security measures introduces security vulnerabilities [14,15]. The latter is HTTP Strict Transport Security (HSTS); HSTS acts as a mechanism for websites to declare themselves accessible only via secure connections. It mitigates SSL stripping attacks by enforcing the use of SSL/TLS in web browsers. Web servers declare HSTS by using HTTP response headers [16,17].
- SMTP: Initially, SMTP communication was sent in clear text, posing security risks. To address this, implicit TLS was introduced. However, it was deprecated for SMTP, and STARTTLS emerged as an alternative protocol for initiating a TLS session. However, it introduces potential vulnerabilities such as STARTTLS stripping attacks [18].
- TCPCrypt: Was developed to ensure end-to-end encrypted communication between applications using TCP. TCPCrypt initiates a secure channel by starting a TCP connection, similar to SSL. However, it lacks the guarantee of authenticity provided by X.509 certificates or passwords, making it susceptible to opportunistic encryption if one endpoint does not support TCPCrypt [19,20].
- DHCP: Despite being widely used, DHCP communication is sent in plain text and lacks authentication mechanisms, creating security vulnerabilities. DHCP snooping is a recognized defence mechanism against starvation attacks, but it does not achieve confidentiality and authenticity [21,22].
- Kerberos: This is one of the most widely used software-based key distribution services. However, it mainly relies on a KDC server that must be available and operating appropriately for security services to succeed. This means that any point of failure will disturb the availability of the whole process. Moreover, root-level access to the KDC server provides the attacker with unrestricted access to the whole system, leading to the compromise of the entire Kerberos database. Plus, Kerberos is not SSL-based, and it does not achieve end-to-end encryption between two clients by itself, which is the basis for secure intra-domain communication [23,24].

Despite the mechanisms mentioned above, the encryption of backend communication channels and the security of intra-domain communications demand further innovation. Novel approaches aim to rectify these gaps by introducing more robust encryption and authentication methods. One novel approach, proposed by Shue et al. in [9], tackles the issue of intra-domain communication by introducing an SSL-based framework to enhance the authenticity of DHCP communication. This framework involves configuring a public-key pair for the local organization's domain and servers. Server certificates, signed with the domain's private key to ensure authenticity, are issued by the DHCP server after verifying users' credentials through a captive portal. Although this approach effectively addresses endpoint authenticity to mitigate DHCP spoofing attacks, it lacks specifications for key exchange algorithms for subsequent communication encryption and fails to implement crucial certificate management practices like revocation, validation, and renewal. In contrast, Microsoft has introduced a recent approach for SharePoint server-to-server authentication within a SharePoint farm [25]. This method involves creating certificates for each SharePoint server, submitting them to a third-party certificate authority for signing, and then importing them back into the Windows Certificate Store on each server. This ensures that whenever a SharePoint server requests information from its peer, the request must be accompanied by the sender's certificate to initiate an SSL handshake and encrypt further communication. Unlike the previous approach, this method includes SSL handshake and key exchange algorithms, and it properly manages certificates through an external certificate authority. The issue with their approach arises from their dependence on an external CA. This dependency introduces higher costs associated with the limited validity period of certificates and the complexities of managing certificates across multiple servers. Furthermore, this approach lacks standardization, as it is specifically tailored to work only with SharePoint servers, limiting its applicability and flexibility in diverse server environments. Another strategy involves the use of "Module-OT", a hardware security module (HSM) developed by the National Renewable Energy Laboratory (NREL) to secure communication between distributed energy resource systems [26]. HSMs are tamper-resistant hardware devices that facilitate end-to-end encryption, authentication, and authorization by generating cryptographic keys and X.509 certificates and performing various cryptographic and certificate management operations. "Module-OT" complies with the Federal Information Processing Standard (FIPS) 140-2, ensuring strict security measures, including preventing unauthorized access, employing up-to-date cryptographic algorithms, and undergoing rigorous testing and evaluation. While this model demonstrates effective data encryption and decryption using symmetric key cryptography, it neglects authentication and SSL encryption, leaving potential security gaps [27].

To summarize the security mechanisms and implementations, Table 1 compares their capability to support peer-to-peer communication, perform encryption, rely on SSL for authentication and encryption, and enforce encryption or terminate communication when configurations are mismatched. These innovative solutions not only address the shortcomings of existing security mechanisms but also underscore the importance of a comprehensive strategy that integrates encryption, authentication, and certificate management to safeguard intra-domain and inter-domain communications effectively.

Table 1. Comparison of security mechanisms for networking protocols and industrial implementations.

Protocol	Security Mechanism/Solution	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
Kerberos	Kerberos	No	No	No	No
DHCP	DHCP snooping	No	No	No	No
	Shue et al. [9]	Yes	No	No	No
-	Microsoft [25]	Yes	Yes	Yes	Yes
-	Module-OT [26]	Yes	Yes	No	No

3. HSM4SSL Architecture

3.1. Inspiration from the Previous Work

After examining current networking protocols and their security structures, as outlined in Table 1, and reviewing existing solutions developed by both academic researchers and industry, it becomes apparent that no existing mechanism offers optimal security for confidential and authenticated communication. Attempting to integrate these mechanisms to secure communication often results in security gaps. Furthermore, existing solutions are often tailored to specific environments or servers, neglecting to leverage all cryptographic operations essential for robust security.

That being said, there is a pressing need for a unified security mechanism within intra-domain environments that integrates SSL with all relevant networking protocols to ensure the security of both web-based and non-web-based communication. However, implementing SSL requires the use of numerous cryptographic keys and the adoption of effective certificate management practices. A commonly employed approach to address this challenge involves the utilization of *key management service* (KMS). Notably, cloud-based KMS solutions, such as Amazon's *AWS KMS* [28], facilitate the generation, storage, management, and distribution of keys and also serve as a certificate authority to perform all necessary certificate management practices. In contrast, HSMs are physical devices installed to securely execute functions equivalent to those of KMS systems. The main differences between HSMs and KMSs include the following:

- *Security*: HSMs provide a high level of security by adhering to the FIPS 140-2 standard, thus making them more reliable. Conversely, a recent exploitation of a vulnerability in Amazon's KMS by Thai Duong [29] resulted in the leakage of clients' private information, among other risks.
- *Flexibility*: KMS systems are generally easily integrated with software programs and services on servers. In contrast, HSMs necessitate the use of vendor-specific libraries for operation, significantly limiting flexibility when substituting an HSM from one vendor for another.
- *Cost*: KMS systems operate on a pay-as-you-go model, often proving to be a cost-effective alternative compared to the deployment of HSMs, which typically entail upfront costs or long-term commitments.

In contrast to software-based alternatives, the security offered by HSMs underscores the necessity of incorporating HSM services in intra-domain environments. However, this highlights a crucial cost-security trade-off, where HSMs offer a significantly higher level of security but at a higher cost compared to cloud-based KMS models. Organizations that manage highly sensitive or critical data are naturally inclined to prioritize investments in robust security mechanisms. Thus, this requires a thorough cost–benefit analysis for organizations considering protecting their sensitive data, taking into consideration the direct costs of acquiring and managing HSMs against the benefits of enhanced security, a reduced risk of data breaches, and compliance with regulatory standards. Moreover, HSMs are available in various designs, each with its own set of features and pricing models, where the selection of HSMs requires a detailed analysis of the organization's specific needs, such as the required cryptographic capabilities, performance specifications, and compliance requirements. Nevertheless, the challenges associated with vendor-specific libraries can pose deployment difficulties when transitioning between HSMs. To address the issues of flexibility linked with HSMs, we propose the development of the *HSM4SSL* architecture.

3.2. HSM4SSL: High-Level Architecture

HSM4SSL is a *software as a service* (SaaS) architecture designed to revolutionize the way organizations implement secure communication within intra-domain environments. This innovative approach combines the power of HSMs with the flexibility and scalability of cloud-based services. HSM4SSL 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, HSM4SSL

ensures the confidentiality, integrity, and authenticity of data exchanged between endpoints. HSM4SSL stands out because it uses the advanced features of HSMs and combines them with other cutting edge technologies to make SSL communications much safer within intra-domain networks. This integration not only maintains the benefits of SSL (such as encryption and data integrity) but also leverages the robust security features offered by HSMs, including enhanced encryption algorithms, secure cryptographic key management, and robust protections against physical threats.

Figure 1 shows an example of the deployment of the HSM4SSL service model provided by a service provider to deliver security services to external client companies. The service provider manages cryptographic operations using HSMs. This secure service hub is responsible for delivering SSL certificate management, encryption, and cryptographic key services to client companies. Each of the three gateways represents a client company connected to the central HSM4SSL service. These companies are subscribed to the service provider's security services to utilize a suite of security services provided in their intra-domain networks. The scalability of this model is one of its key features, as when more client companies subscribe to HSM4SSL services, the system can expand its capacity without compromising the integrity and security of the architecture. This means that, regardless of the number of client companies added or the volume of security transactions processed, the HSM4SSL service maintains consistent security levels, ensuring that each client company's data remain protected.

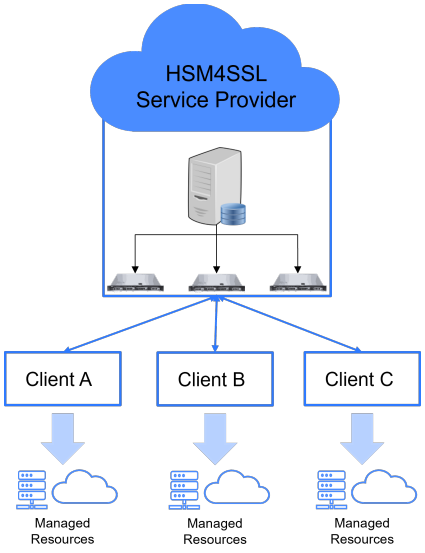

Figure 1. HSM4SSL service model.

Moreover, this architecture offers a zero-touch deployment model, reducing the time and effort required for implementation and allowing organizations to seamlessly integrate HSM4SSL into their existing infrastructures with minimal configuration. HSM4SSL is dedicated to securely generating, storing, and managing cryptographic keys and certificates, as it offers a standardized and user-friendly command-line interpreter (CLI) tool.

The HSM4SSL general architecture, shown in Figure 2, is a three-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, databases, and virtual HSMs, where communication between the physical layer and the upper layers is facilitated by an *adaptive driver communication protocol* (ADCP). 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 HSM4SSL. Finally, the top layer is the application layer, where the SSL protocol and software applications reside. This layer leverages the

underlying software and hardware components through an *application programming interface* (API) to use the various HSM4SSL *security services*. The security services provide a wide range of cryptographic operations and HSM-related functions. Some of the key security services offered by HSM4SSL include the following:

1. Secure storage initialization.
2. Key pair generation.
3. Digital certificate generation.
4. Data encryption/decryption.
5. Access control and authorization.

Moreover, HSM4SSL is a modular architecture designed to provide uniform integration across various network environments and configurations. This adaptability is essential for addressing legacy and modern systems' different security requirements and designs. This is achieved by the usage of uniform CLI commands that provide a consistent interface to eliminate overhead when using HSM4SSL's services in pre-existing infrastructures. Plus, the ADCP parameters can be dynamically adjusted to align with the exact requirements of both advanced and legacy systems, and the HAL can also be effortlessly modified to include HSMs' drivers when more recent HSMs are introduced in the physical layer to fulfill the organization's security requirements.

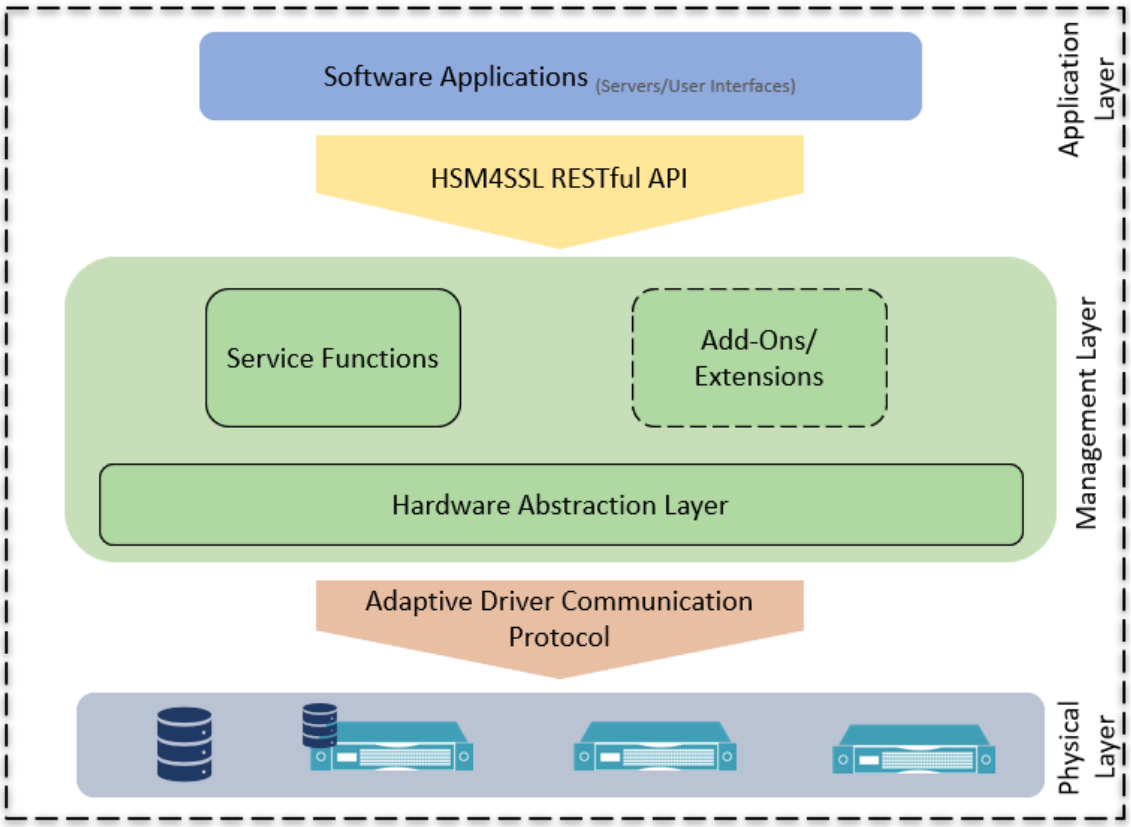

Figure 2. HSM4SSL high-level architecture.

3.3. HSM4SSL: Low-Level Architecture

The low-level architecture of HSM4SSL, shown in Figure 3, offers an in-depth exploration of the internal components and their intricate interactions within the system.

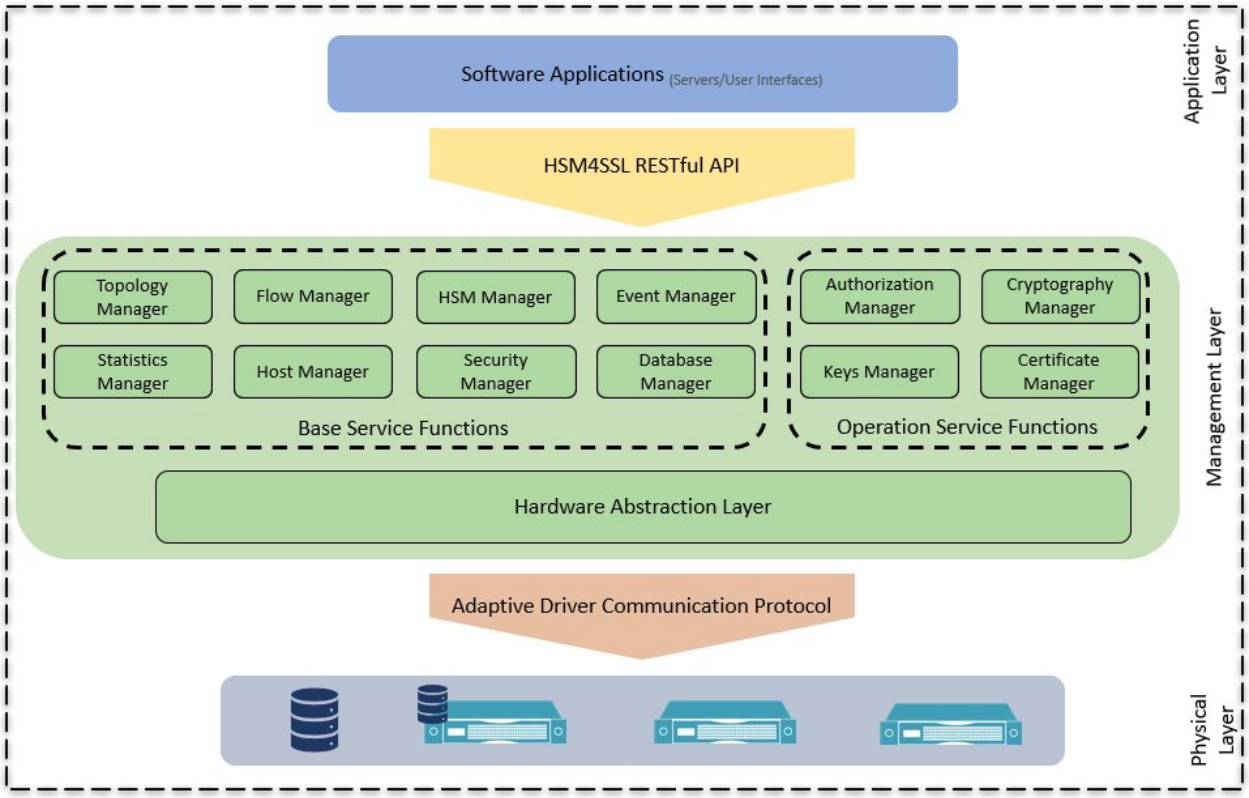

Figure 3. HSM4SSL low-level architecture.

3.3.1. HSM4SSL Application Layer

The application layer of HSM4SSL includes software applications running on intra-domain servers. These servers utilize the security services utilized by HSM4SSL by interacting with the management layer via an API to provide unified and standard access to the available HSMs.

HSM4SSL API: To establish a unified access point for software applications to the architecture, we developed RESTful APIs that adhere to the REST architecture. These APIs utilize HTTP requests to communicate with resources identified by URLs, enabling the exchange of information through standard HTTP methods like GET, POST, PUT, and DELETE. The API design process was complemented by conducting a comparative analysis of the currently most-utilized API development tools, which are FastAPI, Flask, Spring, and Django. Figure 4 presents a performance comparison of the aforementioned tools based on the number of RPS that each framework can handle as the number of concurrent users increases from 0 to 40. From the results, Django demonstrates the highest RPS rate across the board. However, FastAPI preserves a consistent RPS rate, suggesting robust performance under a concurrent user load. Further, the latency comparison shown in Figure 5 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 the additional load with minimal latency impact.

After evaluating the performance of various web frameworks, we chose FastAPI for its impressive balance of handling a high number of requests per second and its 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 a lower latency is crucial for our application, as it directly impacts the responsiveness and efficiency of the service.

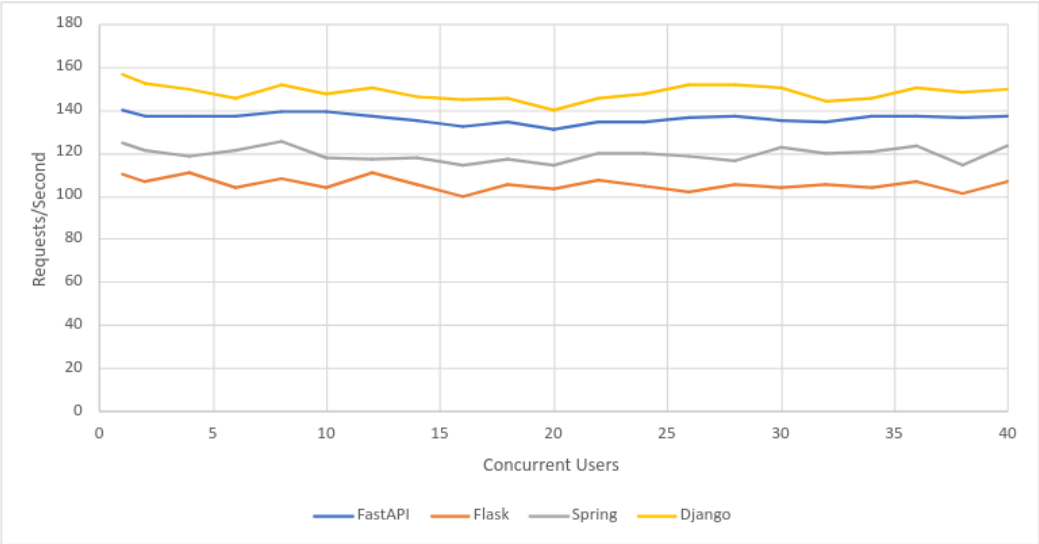

Figure 4. RPS comparison results.

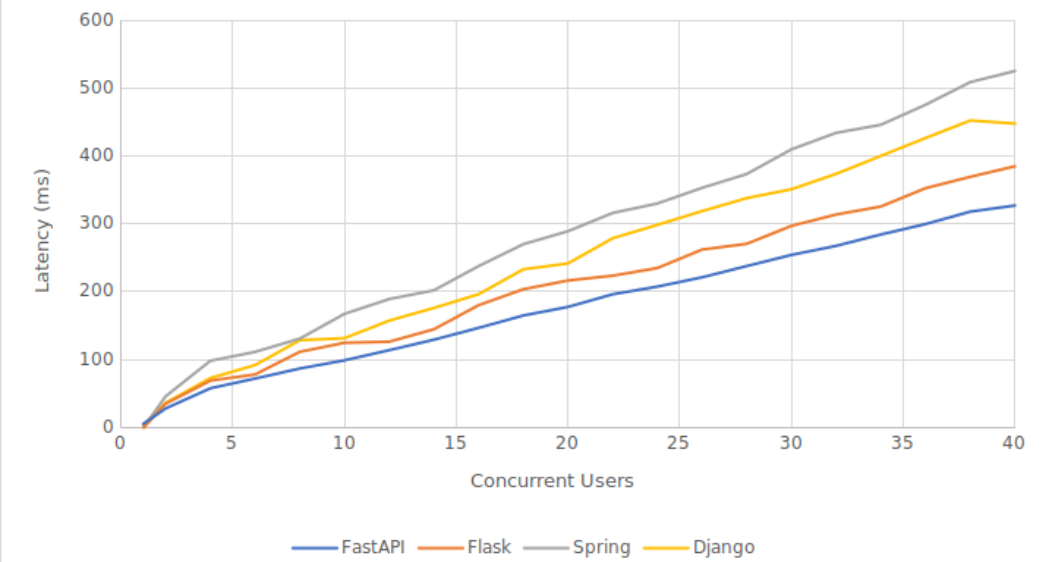

Figure 5. Latency comparison results.

3.3.2. HSM4SSL Management Layer

Exposing the higher-level API to communicate with the underlying HSMs happens in the management layer, where HSM4SSL's centralized control and monitoring mechanisms come into play. This is primarily made up of multiple service managers that can be classified as follows:

1. Base Service Functions:
   The base service functions are employed to provide the primary services required to properly operate the HSM4SSL. The base service functions include the following:
   - 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.
- Host Manager: Manages and stores connected endpoint 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. Maservice managerlearning-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. Moreover, in case of an HSM failure error generated by the security manager, the event manager communicates with the flow manager with a specific error code requiring it to update the policy set. This allows the HAL to route incoming requests to available HSMs until the error is handled.
- Database Manager: Manages and backs up data storage and generates statistics about saved data, such as certificates and audit logs.

2. Operation Service Functions: On the other hand, operation service functions contain managers oriented to perform specific cryptographic and HSM-related tasks, such as the following:
    - 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.
    - 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.

3.3.3. Hardware Abstraction Layer

The HAL, shown in Figure 6, is the heart of the HSM4SSL. It enables software applications to communicate with different HSMs by abstracting the low-level details of the underlying HSMs. It contains a *service manager* and a *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.

The HAL communicates with the flow manager to obtain the policy set. 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 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.

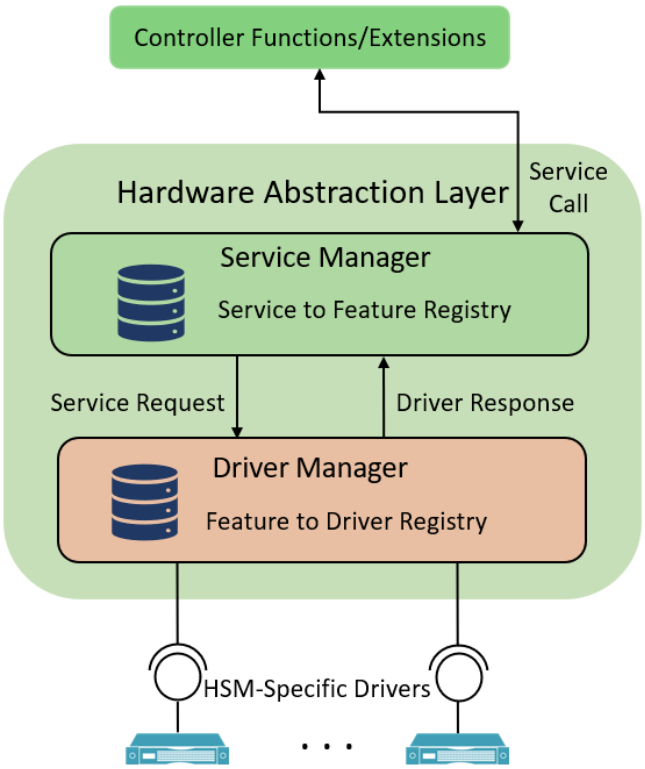

Figure 6. The HSM4SSL hardware abstraction layer (HAL).

When a request is received from the controller managers, the service manager translates the request into a feature request. This translation process entails applying the policy set to ensure the inclusion of the appropriate destination HSM 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 maintains 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.

3.3.4. Adaptive Driver Communication Protocol (ADCP)

Communication between different components is facilitated using the ADCP. The ADCP is a JSON-based messaging protocol that forms the backbone of the communication infrastructure within HSM4SSL. The standardized format, shown in Figure 7, provides a structured approach to exchanging information and instructions. The message is organized into two primary objects: the header and the body.

```
{
    "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_algorithm",
             "days": "certificate_validity"
         }
    }
}
```

Figure 7. The adaptive driver communication protocol (ADCP) standard JSON-based message.

The header object includes keys such as *message_direction*, which denotes if this message is a request for a service or a response to the client. This key can also hold a value to indicate that a message is for internal purposes, such as messages between managers for information exchange. Another key 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"* key indicates the specific HSM operation to be performed. The *"source_mac_address"* key 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''* key denotes the IP address of the source device, providing additional internal architecture usage information. Further, the *"timestamp"* key 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"* key, which specifies the destination HSM where the command should be executed. It identifies the path to the HSM library file responsible for handling the requested operation. By incorporating these standardized header objects, the message format ensures consistency, interoperability, and efficient processing across different HSM-related operations and systems.

The body object of the message format, on the other hand, encapsulates the specific keys and values related to the intended operation or task. Within the body section, additional keys and values that are specific to the operation defined in the header are included. For instance, the *"session_label"* key represents the label or name assigned to the secure session being initialized or an identification of where to store generated cryptographic elements. This allows for the proper identification and management of different sessions within the HSM. The *"credentials"* sub-object contains the *"admin_pin"* and *"user_pin"* keys. 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"* sub-object encompasses keys related to the generation of cryptographic key pairs. The *"type"* key specifies the type of cryptographic algorithm to be used, such as RSA, AES, or EC. The *"length"* key denotes the desired length of the generated encryption key. Additionally, the *"label"* key assigns a unique name or label to the generated public-key pair, facilitating identification and management. Finally, the *"id"* key provides an ID number for the generated public-key pair, enabling easy referencing and retrieval when needed. Moreover, for X.509 certificate generation requests, the *"csr_details"* sub-object 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 HSM4SSL server. This 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 sub-object, *"cert_details"*, is also involved in the certificate generation security service. *"cert_details"* includes two important keys. 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 HSM4SSL administrators.

3.3.5. ADCP Assignment Policy Set

The effective operation of the HAL depends on a well-structured JSON-based assignment policy set. The policy set design is a critical aspect of the HSM4SSL architecture, as it determines the proper assignment 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, which ensures efficient resource utilization and optimal performance. For example, the policy set shown in Figure 8 includes the *"hsm"* key, which identifies the specific HSM to which the policy applies. For each HSM, the *"capabilities"* key 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 public-key pair generation, certificate generation, and encryption, while HSM2 can only perform key pair generation and encryption. Plus, the *"algorithms"* key 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"* key 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"* key 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 assignment policy set, the HSM4SSL architecture can effectively optimize resource allocation and ensure compatibility with cryptographic operations.

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

Figure 8. HSM4SSL JSON-based assignment policy set.

3.3.6. HSM4SSL Physical Layer

The physical layer includes the hardware components and infrastructure necessary for the secure operation of the system, such as the following:

- HSMs: The physical layer includes tamper-resistant HSMs that provide cryptographic services. HSM4SSL relies on the security of those HSMs as they adhere to strict security standards such as FIPS 140-2, which are engineered to protect sensitive data and manage cryptographic keys with unmatched security, thereby strengthening the SSL operations, which are crucial for secure intra-domain communication. Plus, to accommodate various needs and requirements and to allow for greater flexibility, HSM4SSL supports an array of HSMs from multiple vendors. This multi-vendor approach ensures that organizations are not locked into a single provider, thereby reducing the risk of vendor lock-in and allowing for a customized security fit. The integration process is streamlined, ensuring that adding a new HSM to the environment is as simple as possible. Moreover, in light of the evolving landscape of cybersecurity threats due to the arrival of quantum computers, HSM4SSL must evolve at a comparable pace. Of note is that the National Institute of Standards and Technology (NIST) has selected four cryptographic algorithms covering key exchange and digital signatures that are pending standardization. However, the development of post-quantum safe HSMs is considered a hot topic by leading HSM manufacturers, such as Thales and Crypto4A [30,31], as building a post-quantum safe HSM requires completely different cryptographic frameworks compared to those employed in contemporary HSMs. Moreover, as HSM4SSL is designed for quick adaptability, we are certain that as these post-quantum safe algorithms and HSMs are standardized and developed, HSM4SSL will be able to incorporate these HSMs into the architecture.
- 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.
- Virtual HSM: In addition to physical HSMs, the physical layer supports the virtual HSM, also known as a HSM simulator. It is important to note that the HSM4SSL architecture has been developed and thoroughly tested in the SoftHSM simulator environment. SoftHSM provides a software-based emulation of an HSM and offers a controlled environment for cryptographic operations and secure storage [32].

The physical layer is designed with scalability in mind. As new cryptographic challenges arise and industrial needs change, HSM4SSL's physical layer can easily incorporate newer HSMs that meet these evolving standards. Plus, the HAL can be dynamically updated to include the newly added HSM's driver to facilitate communication. This process allows HSM4SSL to leverage the latest HSM technology without requiring any modifications to the system's core architecture and ensure long-term viability and effectiveness.

3.3.7. HSM4SSL CLI Tool

The developed *"ssleverywhere"* command-line interpreter (CLI) tool serves as a versatile interface for interacting with the HSM4SSL 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 HSM4SSL 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 HSM4SSL, be signed, and create X.509 certificates. Finally, the *"certs"* folder stores the issued certificates obtained after successful CSR submissions. 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 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.

Another command is *"-keypair_gen"*. This command is used to request HSM4SSL 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 HSM4SSL database.
- *-key_type*: Specifies the type of cryptographic key to be generated.
- *-length*: Specifies the desired length of the key.

The usage example of the *ssleverywhere* tool involves two machines on the same network establishing secure communication. Machine A, the server, generates a certificate named "SE" using the *ssleverywhere* command:

```
$ 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
```

Machine B, the client, follows a specific configuration to generate its certificate:

```
$ 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
```

Machine B generates a certificate named "CL" with specific configurations: RSA key type, 4096-bit key length, DES3 encryption, and SHA-512 hashing algorithm. After successful generation, both machines have their certificates, enabling secure communication.

Both machines possess certificates from the HSM4SSL architecture, as per their specific requirements. To utilize these certificates, Machine A acts as the server, employing a Python socket programming code to listen for connections on a predetermined port. It loads its SSL certificate and the root CA certificate into the SSL/TLS server context, awaiting client connections. Machine B initiates a connection to Machine A's server using its own SSL certificate and the root CA certificate in its SSL/TLS client context. The SSL handshake process occurs upon connection establishment, facilitating mutual authentication. Once completed, a secure channel is established, allowing encrypted and integrity-protected application data exchange between the client and server. With both programs running and configurations in place, the SSL handshake process, as captured by Wireshark and illustrated in Figure 9, 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 the client and server form a secure channel and can exchange application data securely, which is encrypted and integrity-protected.

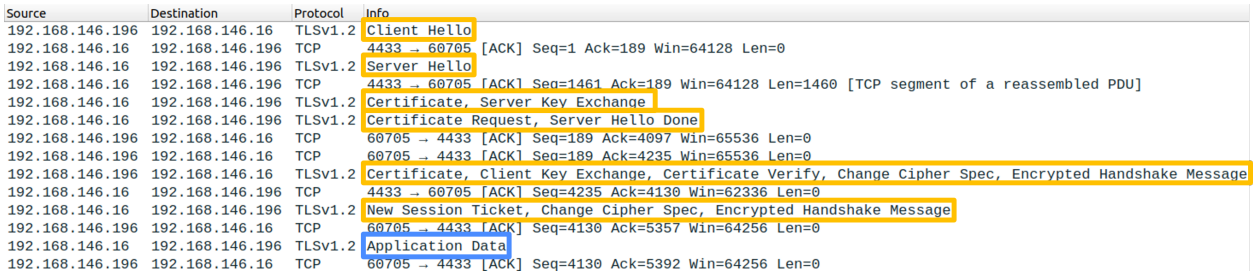

Figure 9. Captured handshake process.

4. HSM4SSL Performance

To perform a valid performance evaluation of the HSM4SSL architecture where three HSM simulators are deployed, it was compared with *OpenSSL* version 3.0.11, 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 [33]. Note that the sole usage of *OpenSSL* in sensitive intra-domain environments being 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 to 2019 [34].

The experimental setup involved testing on two separate PCs within the same network environment to ensure a consistent and controlled testing framework. The primary machine, acting as the server and used for hosting the HSM4SSL architecture, had the specifications detailed in Table 2. The secondary machine, responsible for generating certificate requests and performing benchmarks, was equipped to emulate realistic operational conditions. The simulated environment used SoftHSM version 2.6.1, an open-source software implementation of a cryptographic store accessible through a PKCS#11 interface. This setup allowed for the testing of the HSM4SSL architecture's integration with HSM capabilities and without the need for physical hardware. SoftHSM was configured with the default settings, such as its support for ECC, AES, and RSA for cryptographic algorithms,

and non-paged memory was maintained in its enabled state to prioritize security over performance, ensuring that sensitive data were kept out of disk-based swap spaces, thus minimizing the risk of data leakage. Plus, P11-kit integration was enabled, facilitating the interaction of SoftHSM with PKCS#11 modules installed on the system to specify how to allow the user to communicate with the cryptographic tokens. Moreover, SoftHSM uses the Botan cryptographic library by default for its cryptographic operations.

The network connecting these devices was configured with a bandwidth of 127 MB/s and a latency of 5 ms between the two devices, mimicking intra-domain network conditions that might impact performance metrics.

Table 2. Hardware specifications.

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

As for the software tools, we used the *loadtest* command line and Node.js module version 8.0.3, which simulates high volumes of traffic to web applications to assess their performance, identify bottlenecks, and understand how they handle various levels of load [35]. It operates in various modes, such as by specifying the requested 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. The *loadtest* tool was used to simulate real-world high traffic levels and interaction with the HSM4SSL architecture to understand how it scales under stress. Additional tools, such as the *top* (table of processes), which provides real-time insight into active processes and presents a system information view that includes resource utilization data, and *nload*, which monitors real-time network traffic, were employed to monitor performance metrics beyond what loadtest offers.

4.1. Testing Scenarios and Metrics

To ensure precise evaluation, a separate PC was used to carry out all test scenarios using *loadtest*. These scenarios mainly involved varying two parameters, which are the number of concurrent users, which refers to the number of virtual users that are actively making requests to the target application or API at the same time, and the number of requests, which indicates the total number of HTTP requests that will be generated and sent to the API. Moreover, the key evaluation metrics used are the effective RPS, which measures the rate at which the system can handle incoming certificate generation requests; latency, which refers to the time delay or the elapsed time between the initiation of a request and the moment when a response is received; CPU usage, to measure the amount of resources consumed by the CPU during the test; and network bandwidth, which measures the amount of data transmitted over the network during the load test.

4.2. Results and Discussion

Figure 10 shows that both HSM4SSL and OpenSSL have a roughly steady and consistent RPS rate as the number of concurrent users increased from 1 to 100, suggesting that both systems demonstrated an ability to handle an increasing workload without a significant drop in RPS, emphasizing their scalability under the test conditions. However, HSM4SSL consistently exhibits a higher RPS compared to OpenSSL at various user load levels. This indicates the remarkable efficiency and scalability of HSM4SSL in processing a greater number of requests in parallel. Plus, Figure 11 reveals that, as the number of concurrent users increases in the same range, both HSM4SSL and OpenSSL exhibit a gradual increase in latency, which is expected as system loads intensify. However, on

average, the latency for each request served by OpenSSL tends to be 13.08% higher than that of HSM4SSL.

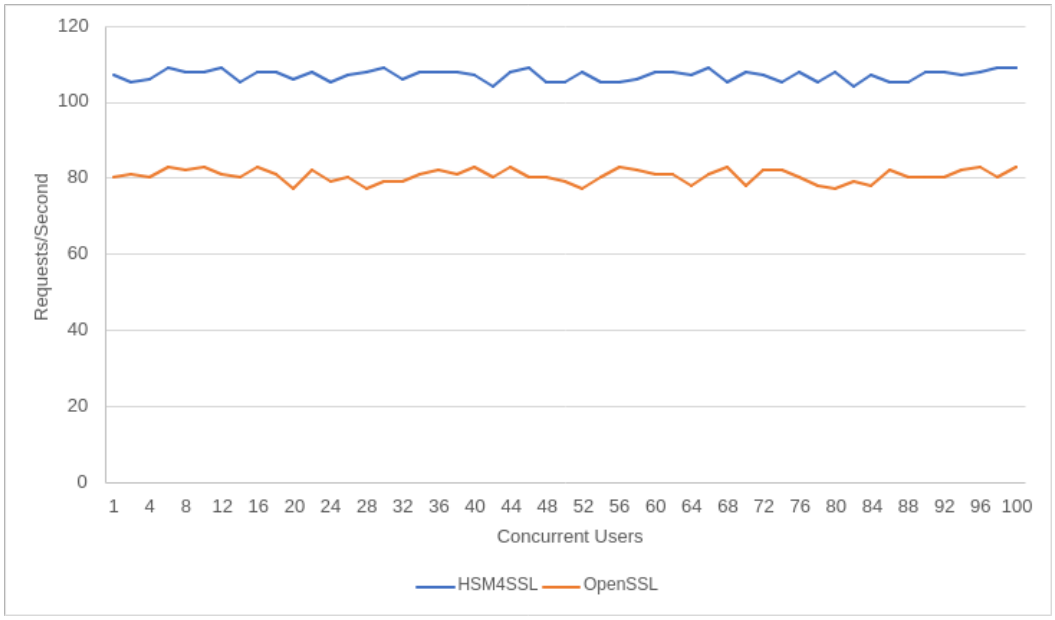

Figure 10. RPS comparison results.

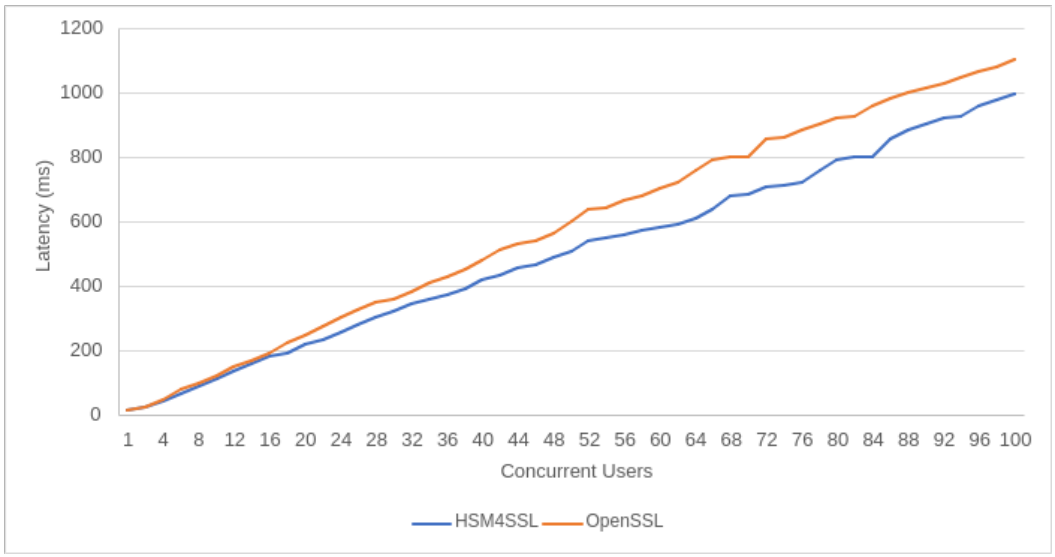

Figure 11. Latency comparison results.

Another evaluation criterion is the evaluation of resource and network utilization, where CPU usage was determined by closely monitoring essential components, namely Python, Google Chrome, and the MySQL database, for the HSM4SSL architecture. An interesting observation from the results shown in Figure 12 is the clear and consistent logarithmic growth exhibited by HSM4SSL and OpenSSL. However, HSM4SSL's growth in CPU usage is more pronounced compared to OpenSSL since HSM4SSL 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, HSM4SSL 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 15.6%. This is because the application or

processes included in HSM4SSL may be configured with resource allocation limits that restrict its ability to consume more than a certain percentage of the CPU's total capacity.

The network utilization concentrates on understanding the influence of certificate generation on network resources, specifically focusing on HSM4SSL. Notably, OpenSSL operates and is utilized locally within each server, making the monitoring of network utilization primarily relevant to HSM4SSL. Figure 13 shows a consistent upward trajectory in network bandwidth consumption by HSM4SSL. However, it is noteworthy that the network bandwidth utilization eventually stabilizes at approximately 690–700 KB/s. This cap indicates that HSM4SSL, while efficiently scaling to meet growing demands, has an upper limit in terms of network resource utilization. This stability underscores the architecture's ability to effectively allocate and manage network resources without excessive saturation.

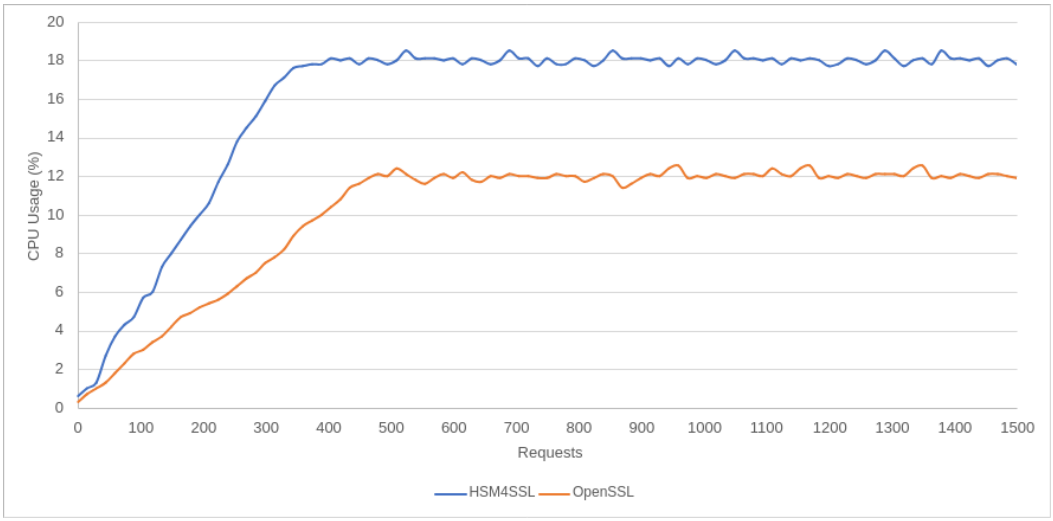

Figure 12. CPU usage comparison results.

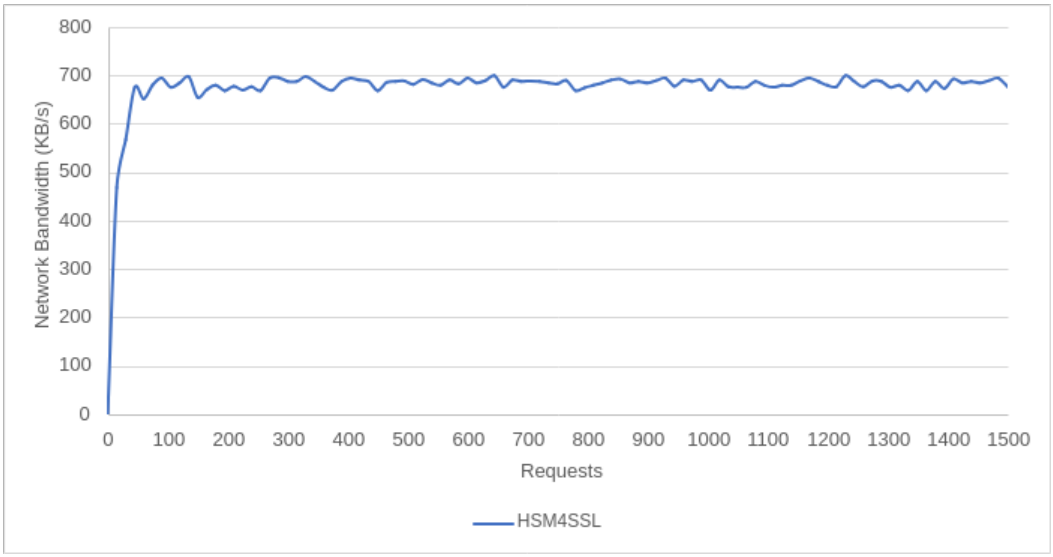

Figure 13. Network bandwidth comparison results.

Based on the evaluation outlined, multiple methods can be involved to enhance the performance of HSM4SSL. The evaluation of HSM4SSL shows a weakness in CPU usage. To reduce CPU utilization caused by MySQL processes, one strategy is to implement database-caching mechanisms. Caching frequently accessed data can reduce the computational load on the CPU by minimizing the need to repeatedly retrieve data from the database. Plus, by utilizing more parallelization techniques, HSM4SSL can reduce the latency and processing

time by enabling the simultaneous execution of multiple certificate generation requests. Moreover, by performing request throttling to limit the number of requests from a single user, HSM4SSL 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.

5. Conclusions

This paper addresses the current security issues for communication in intra-domain environments, as most peer-to-peer communication is unencrypted and unauthenticated, and proposes the HSM4SSL architecture, which 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 insights into its capabilities and contributions.

HSM4SSL is a unified and standardized architecture that aims to strengthen the security of intra-domain environments. Its 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. Plus, comparing HSM4SSL with the widely used OpenSSL demonstrates the former's superior ability to manage requests more efficiently, with higher requests per second and lower latency while ensuring the effective utilization of CPUs and network resources. This performance advantage underscores HSM4SSL's potential to serve as a reliable foundation for secure communication within intra-domain environments. However, it is important to mention that the provided comparative analysis with OpenSSL represents preliminary results, which were aimed at providing a foundational overview of HSM4SSL's performance against a well-established tool within the security realm. Looking forward, we acknowledge the importance of extending our comparative analysis to include a broader range of security tools and technologies, where we plan to perform a comprehensive evaluation that not only revisits our preliminary findings with OpenSSL but also encompasses additional comparisons with other cryptographic tools. Plus, it is crucial to focus on a comprehensive security analysis of HSM4SSL against relevant architectures, including a detailed evaluation of the API in the face of well-known security attacks, along with developing threat models that enable us to simulate various attack victors and assess the system's response under different threat scenarios.

However, HSM4SSL introduces challenges such as the dependency on physical HSMs, which introduces cost complexities, especially when multiple HSMs are deployed. Plus, the architecture, while optimized for multiple HSM integration, may encounter latency and overhead, especially when integrating a single HSM within the premises. Furthermore, despite efforts to ensure system robustness, certain components could become failure points if not managed properly, such as the central management of HSMs, the API, and the centralized policy management system. Additionally, the necessity to regularly update both the HSMs and software components to align with evolving security standards underscores our commitment to continual improvement and adaptation.

In summary, HSM4SSL represents notable progress in tackling security issues in intra-domain situations. However, there are various areas for future study and development that might improve its capabilities and influence. Further research might be conducted to improve the compatibility of HSM4SSL with a wider array of HSMs, therefore enabling enterprises to select from a varied selection of HSM alternatives while upholding the security requirements and principles of the architecture. This presents an opportunity to investigate sophisticated load-balancing methodologies, caching methods, and parallel processing strategies in order to minimize latency and enhance throughput. Furthermore, future endeavors may encompass initiatives to advocate HSM4SSL as a widely accepted option for ensuring secure communication within a certain domain. The attainment of widespread

acceptance may necessitate collaboration with pertinent standards bodies and industry groups. For the time being, HSM4SSL codes and API are under development to meet the standards of security, functionality, and scalability, where the aim is to provide comprehensive documentation, including detailed guides on endpoints, usage examples, and configuration instructions, to support future exploration and validation of "HSM4SSL"'s capabilities.

Author Contributions: Conceptualization, Y.A. and A.O.; methodology, Y.A. and A.O.; software, Y.A.; validation, Y.A. and A.O.; formal analysis, Y.A. and A.O.; investigation, Y.A. and A.O.; resources, Y.A. and A.O.; data curation, Y.A. and A.O.; writing—original draft, Y.A.; writing—review and editing, Y.A. and A.O.; project administration, A.O.; funding acquisition, A.O. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The data is not accessible to the public due to its involvement in an ongoing study.

Acknowledgments: During the preparation of this work the authors used Grammarly tool in order to check the English grammar of the manuscript. After using this tool, the author(s) reviewed and edited the content as needed and take full responsibility for the content of the publication.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Zhang, M. *On the State of the Inter-Domain and Intra-Domain Routing Security*; University of Oregon: Eugene, OR, USA, 2016; p. 23.
2. Schulze, H. 2020 Insider Threat Report. Available online: https://www.cybersecurity-insiders.com/wp-content/uploads/2019/11/2020-Insider-Threat-Report-Gurucul.pdf (accessed on 22 April 2024).
3. Bourke, T. *Server Load Balancing*, 1st ed.; O'Reilly: Beijing, China; Sebastopol, CA, USA, 2001.
4. Membrey, P.; Hows, D.; Plugge, E. SSL Load Balancing. In *Practical Load Balancing*; Apress: Berkeley, CA, USA, 2012. [CrossRef]
5. SectigoStore. SSL Certificate for IP Address—An Expert Guide on SSL for IP Address. Available online: https://sectigostore.com/page/ssl-certificate-for-ip-address/ (accessed on 22 April 2024).
6. Sectigostorepages. Google Announces Intentions to Limit TLS Certificates to 90 Days: Why Automated CLM Is Crucial. Available online: https://www.sectigo.com/resource-library/google-announces-intentions-to-limit-tls-certificates-to-90-days-why-automated-clm-is-crucial (accessed on 22 April 2024).
7. Aref, Y.; Ouda, A. Still Computers Networking is Less Secure Than It should be, Causes and Solution. In Proceedings of the 2023 International Symposium on Networks, Computers and Communications (ISNCC), Doha, Qatar, 23–26 October 2023; pp. 1–8. [CrossRef]
8. Herzberg, A.; Hollick, M.; Perrig, A. Secure Routing for Future Communication Networks (Dagstuhl Seminar 15102). *Dagstuhl Rep.* **2015**, *5*, 28–40. [CrossRef]
9. Shue, C.A.; Kalafut, A.J.; Gupta, M. A Unified Approach to Intra-domain Security. In Proceedings of the 2009 International Conference on Computational Science and Engineering, Vancouver, BC, Canada, 29–31 August 2009. [CrossRef]
10. Boisrond, P.D. To Terminate or Not to Terminate Secure Sockets Layer (SSL) Traffic at the Load Balancer. *arXiv* **2020**, arXiv:2011.09621.
11. Rescorla, E. RFC2818; HTTP over TLS. Available online: https://www.rfc-editor.org/info/rfc2818 (accessed on 22 April 2024).
12. W3Techs. Usage Statistics of Default Protocol Https for Websites. Available online: https://w3techs.com/technologies/details/ce-httpsdefault (accessed on 22 April 2024).
13. Chordiya, A.R.; Majumder, S.; Javaid, A.Y. Man-in-the-Middle (MITM) Attack Based Hijacking of HTTP Traffic Using Open Source Tools. In Proceedings of the 2018 IEEE International Conference on Electro/Information Technology (EIT), Rochester, MI, USA, 3–5 May 2018. [CrossRef]
14. Fung, A.P.H.; Cheung, K.W. 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, 13–16 April 2010. [CrossRef]
15. Chang, L.; Hsiao, H.C.; Jeng, W.; Kim, T.H.J.; Lin, W.H. Security Implications of Redirection Trail in Popular Websites Worldwide. In Proceedings of the 26th International Conference on World Wide Web, Perth, Australia, 3–7 April 2017. [CrossRef]
16. Hodges, J.; Jackson, C.; Barth, A. HTTP Strict Transport Security (HSTS). RFC 6797 (Proposed Standard). Available online: https://www.rfc-editor.org/info/rfc6797 (accessed on 22 April 2024).
17. Dolnak, I.; Litvik, J. Introduction to HTTP security headers and implementation of HTTP strict transport security (HSTS) header for HTTPS enforcing. In Proceedings of the 2017 15th International Conference on Emerging eLearning Technologies and Applications (ICETA), Stary Smokovec, Slovakia, 26–27 October 2017. [CrossRef]

18. Poddebniak, D.; Ising, F.; Böck, H.; Schinzel, S. Why TLS is better without STARTTLS: A Security Analysis of STARTTLS in the Email Context. In Proceedings of the 30th USENIX Security Symposium (USENIX Security 21), Virtual Event, 11–13 August 2011.
19. Bittau, A.; Hamburg, M.; Handley, M.; Boneh, D. The case for ubiquitous transport-level encryption. In Proceedings of the 19th USENIX Security Symposium (USENIX Security 10), Washington, DC, USA, 9 August 2010.
20. Nikolidakis, S.A.; Giotsas, V.; Georgakakis, E.; Vergados, D.D. Towards Utilizing Tcpcrypt in Mobile Healthcare Applications. In Proceedings of the Wireless Mobile Communication and Healthcare, Paris, France, 21–23 November 2012; Springer: Berlin/Heidelberg, Germany, 2012.
21. Bhushan, B.; Sahoo, G.; Rai, A.K. Man-in-the-middle attack in wireless and computer networking — A review. In Proceedings of the 2017 3rd International Conference on Advances in Computing, Communication & Automation (ICACCA) (Fall), Dehradun, Indian, 15–16 September 2017. [CrossRef]
22. Odom, W. *Ccna 200-301 Official Cert Guide*, 1st ed.; Pearson Education, Inc.: Hoboken, NJ, USA, 2019; Volume 2.
23. Motero, C.; Higuera, J.R.; Bermejo, J.; Montalvo, J.A.; Gomez, N. On Attacking Kerberos Authentication Protocol in Windows Active Directory Services: A Practical Survey. *IEEE Access* **2021**, *9*, 109289–109319. [CrossRef]
24. Deland-Han. Kerberos Authentication Troubleshooting Guidance—Windows Server. Available online: https://learn.microsoft.com/en-us/troubleshoot/windows-server/windows-security/kerberos-authentication-troubleshooting-guidance (accessed on 22 April 2024).
25. SerdarSoysal. Authentication Overview for SharePoint Server—SharePoint Server. Available online: https://learn.microsoft.com/en-us/sharepoint/security-for-sharepoint-server/authentication-overview (accessed on 22 April 2024).
26. Han, J.; Kim, S.; Kim, T.; Han, D. Toward Scaling Hardware Security Module for Emerging Cloud Services. In Proceedings of the 4th Workshop on System Software for Trusted Execution, Huntsville, ON, Canada, 27–30 October 2019. [CrossRef]
27. Hupp, W.; Hasandka, A.; de Carvalho, R.S.; Saleem, D. Module-OT: A Hardware Security Module for Operational Technology. In Proceedings of the 2020 IEEE Texas Power and Energy Conference (TPEC), College Station, TX, USA, 6–7 February 2020. [CrossRef]
28. Vereecke, A. Koninklijke Militaire School: Amazon-KMS. Available online: https://aws.amazon.com/kms/ (accessed on 22 April 2024).
29. NIST. CVE-2020-8897 Detail. Available online: https://nvd.nist.gov/vuln/detail/CVE-2020-8897 (accessed on 22 April 2024).
30. Group, T. Post-Quantum Cryptography Solutions. Available online: https://cpl.thalesgroup.com/encryption/post-quantum-crypto-agility (accessed on 22 April 2024).
31. Crypto4A. Available online: https://crypto4a.com/news/quantum-safe-secure-manufacturing/ (accessed on 22 April 2024).
32. OpenDNSSEC. SoftHSM. Available online: https://www.opendnssec.org/softhsm/ (accessed on 22 April 2024).
33. Tomita, C.; Takita, M.; Fukushima, K.; Nakano, Y.; Shiraishi, Y.; Morii, M. Extracting the Secrets of OpenSSL with RAMBleed. *Sensors* **2022**, *22*, 3586. [CrossRef] [PubMed]
34. Walden, J. The Impact of a Major Security Event on an Open Source Project: The Case of OpenSSL. Available online: http://arxiv.org/abs/2005.14242 (accessed on 22 April 2024).
35. NPM. "loadtest" Linux Tool. Available online: https://www.npmjs.com/package/loadtest (accessed on 22 April 2024).