EXCEL JOURNAL OF ENGINEERING TECHNOLOGY AND MANAGEMENT SCIENCE
(An Peer Reviewed International Multidisciplinary Journal) *ISSN 2249-9032 (Print)*
Vol. I No.28 - June 2025 *ISSN 2277-3339 (Online)*
Impact Factor 5.136 (IIFS)

Comparative Performance Evaluation of Transport and Message-Level Plain Text Data Processing Using Intermediary Relay Web Services

**** Dr. Manish L. Jivtode***

Abstract

In today's distributed systems, intermediary relay web services play an important role in facilitating smoothly communicate between web service clients and servers. Here, performance is crucial, especially when handling plain text data, which is simple and widely used. This study presents a comparative performance of transport-level and message-level plain text data handling using intermediary relay web services by testing key parameters such as efficiency, speed, and scalability under various conditions. Using a combination of simulation and real-world tests, the study examine how factors such as data size, network conditions, and multiple requests impact overall system performance. The findings will help improve relay web services for faster and more reliable in real-world applications.

Keywords: Distributed system, Relay web services, Plan text data, Transport-level, Message-level, Efficiency, Speed, Scalability, Network condition.

1. Introduction

The security of web services is critical for ensuring message integrity, confidentiality, privacy, and authentication in both client-to-server requests and server-to-client

* Head & Assistant Professor, Department of Computer Science, Janata Mahavidyalaya, Chandrapur

responses. To achieve these goals, a variety of authentication mechanisms are used, including Windows authentication, password-based authentication, X.509 digital certificates, custom authentication modules, and issued tokens. Additionally, digital signatures and message verification techniques are employed to maintain message integrity and authenticity.

In this context, intermediary relay web services are utilized to assess and enhance the security posture of RESTful web services by implementing both Transport Layer Security (TLS) and Message-Level Security (MLS). Although modern REST web services typically depend on TLS, message-level security provides more granular and flexible control.

The fundamental distinction between TLS and MLS lies in their approach to securing messages. TLS provides point-to-point security by encrypting the entire communication channel between two nodes. Once the message exits this secure channel such as when reaching an intermediary, the message is decrypted, making it potentially vulnerable. In contrast, MLS secures the message itself by embedding credentials, claims, and encrypted data within the message structure. This allows for end-to-end confidentiality, ensuring that sensitive information remains encrypted until it reaches the intended recipient, even across multiple intermediaries.

Message-level security offers several advantages over transport-level security. It supports multi-layered encryption, allowing different sections of a message to be encrypted using distinct algorithms or keys. This enables fine-grained access control, where specific message parts are visible only to designated recipients. Furthermore, MLS permits unencrypted routing information to be included in the message, enabling intermediary services to process or route the message without compromising the confidentiality of sensitive data.

The motivation for adopting message-level security over traditional transport security lies in its ability to deliver flexible, content-aware, and recipient-specific protection. Different parts of a message can be encrypted using different credentials, allowing a single message to be targeted to multiple audiences. Additionally, routing metadata can

remain visible to intermediary services while ensuring that the message body remains confidential.

Figure (1) - Implementation of Transport Level Security in REST ful web services

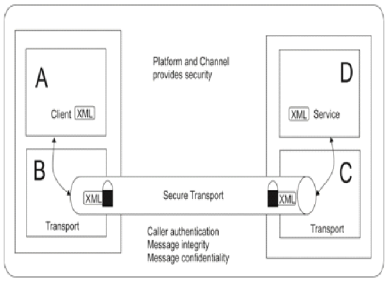


Figure - 1 illustrates a REST web service architecture secured using transport-level security. In this configuration, "**A"** represents the client, "**D"** is the REST service, and "**B"** and '**C'** represent the transport channel or communication pipeline. Sensitive data is encrypted within the channel, but it is automatically decrypted once it exits the pipeline at any intermediary point, such as "B" or "C".

2. Literature Review

In today's digital world, web services are commonly used to share data between different systems through the internet. When a large amount of data is sent online, it is important to make sure the data reaches safely and remains protected.

Many times, data is not sent directly. Instead, it goes through intermediary services, like relay servers or gateways, which help forward or process the data. These services can affect both the speed (performance) and safety (security) of data transmission.

This literature review will look at past research and studies related to web services, security methods, intermediary services, and how all these factors affect the performance of plain text data processing.

Muthukrishnan (2020) conducted a technical analysis on security realization in web services for e-business management. Their research focused on the challenges of implementing security measures in distributed computing environments, particularly when intermediary services are involved.

Choudhary R.K. (2013) proposed a security model based on the HTTPI protocol for SOAP-based web services. Their implementation demonstrated improved performance metrics, such as higher throughput and lower response times, compared to traditional HTTPS-based security approaches.

Gandhi and Dhabaria (2012) provided a comprehensive review of various message security services using XML. They discussed the implementation of XML Signature and Encryption techniques to ensure the confidentiality and integrity of messages in service-oriented architectures.

Makino (2004) evaluated the performance of WS-Security in real-world middleware implementations. Their study revealed that **intermediary services can degrade system performance** when full message-level security is applied, especially with large XML payloads. They emphasized the importance of **balancing security needs with processing capacity**, particularly in systems that involve multiple web service hops or relay nodes.

Objectives of the Study

The main objective of this research is to evaluate and compare the performance of Transport level and Message level security approaches in plain text data processing using intermediary relay web services. This comparison focuses on aspects such as message confidentiality, integrity, speed and scalability in distributed cloud-based environments.

a) The Problem

Transport Layer Security (TLS) is widely used for securing RESTful web service communications; however, it has notable limitations, particularly in systems involving intermediary components. The following observations highlight key considerations-

1) Use transport layer security (TLS) only if, client is sending a message directly to REST web service with no intermediary system is present in between REST service and client.
2) **If an intermediary system is present**, such as a relay web service, TLS does

not provide end-to-end security. In such cases, message-level security (MLS) is preferred to ensure confidentiality and integrity.

3) **TLS is appropriate** in trusted, closed environments such as intranets where both the client and service reside within the same secure network
4) **In public internet environments**, especially where intermediary relay services are used, MLS is recommended to ensure robust and persistent protection of sensitive data

In intermediary scenario, the message itself is not protected once an intermediary reads from the wire and must be retransmitted to the ultimate receiver in out-of-band fashion, if necessary. This applies even if the entire route uses SSL security between individual hops.

Figure (2) – Message Level Security in REST web services

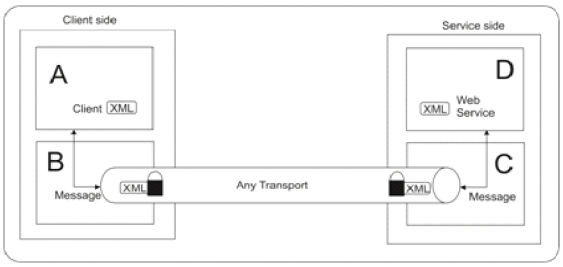


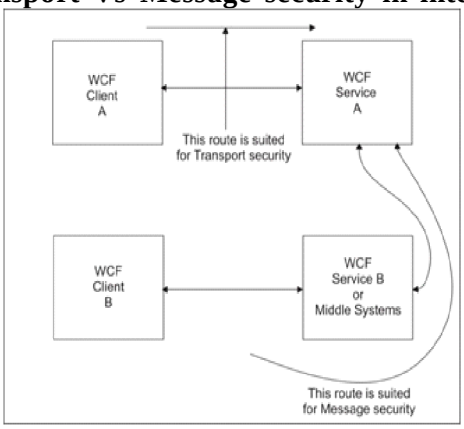
Figure (2) shows message level security architecture in REST web services where A is client, D is REST service, B and C is the transport channel or pipe. Sensitive data is protected within and outside the channel or pipe also. When data passes the channel or pipe at B and C, it remains encrypted until it reaches its final destination.

b) The Solution

Message level security has following known facts -

1) Use message level security when an intermediary is present to provide end-to-end security.
2) Use message level security in an internet environment.
3) To have a fine grained control over the message parts.
4) Support for multiple transports such as named pipe, TCP (not possible with REST because it is based only on HTTP).
5) Support for a wide set of credentials and claims.

Figure (3) - Transport Vs Message security in intermediary environment



c) Reasons for using Intermediary Relay Web Service

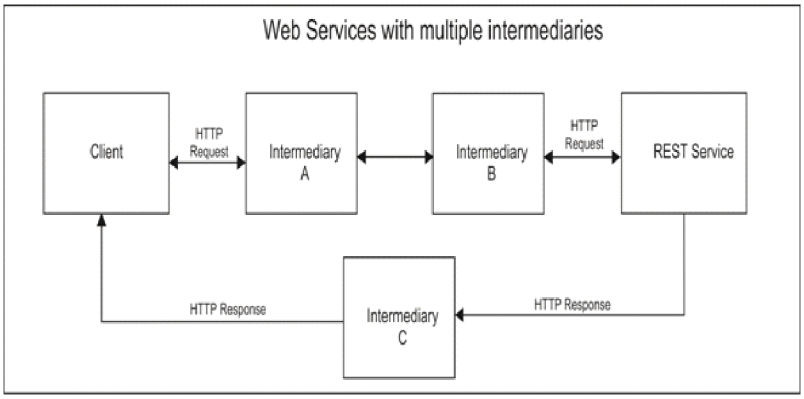
Following reasons for using intermediary relay web services –

1) Actual web service is the first directly web service exposed to the outside world attacks and to protect the actual web service from outside attacks.
2) There may be a group of web services rather than only one web service.
3) Intermediary relay web service used as a message router.
4) There may be multiple organizations involved in the financial transactions like e-shopping sites etc. This will require at least one web service per organization and hence a group of web services.

Intermediary Architecture

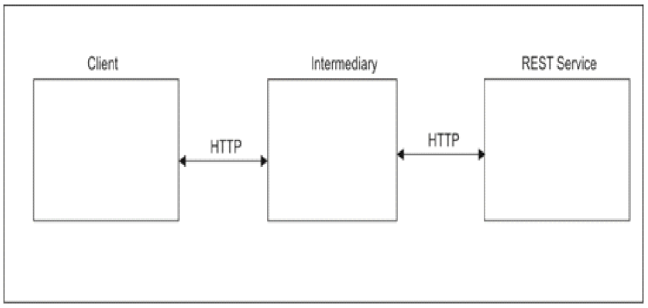
An intermediary is a component that lies between the client and the actual service. When the message is sent from the initial sender, it may pass through intermediate nodes before reaching its intended receiver. It basically intercepts the request from the client, routes it to the correct actual final web service. Similarly, it may intercept the response from the actual final web service and forward it to the client. Figure 4.4 shows an intermediary web service between the client and actual web service. It intercepts the client requests and forwards it to the actual web service.

Figure (4) - Intermediary between client and service during request



In the above diagram, it is possible to combine intermediaries in several ways. In below figure (5), a chain of intermediaries A and B intercepts the request from the client. Another intermediary C intercepts the response from the actual web service.

Figure (5) - Intermediary between client and service during request and response



These intermediaries are increasingly getting recognized as the means to provide value added services like authentication, quality of service (QoS), auditing, management, aggregation etc." Jivtode M. (2020)

To study the transport layer and message level security in REST web services, request and response messages between web service and clients can be inspected using one method:

Design and implement a REST web service and REST client with intermediary REST relay web service.

Experimental Approach

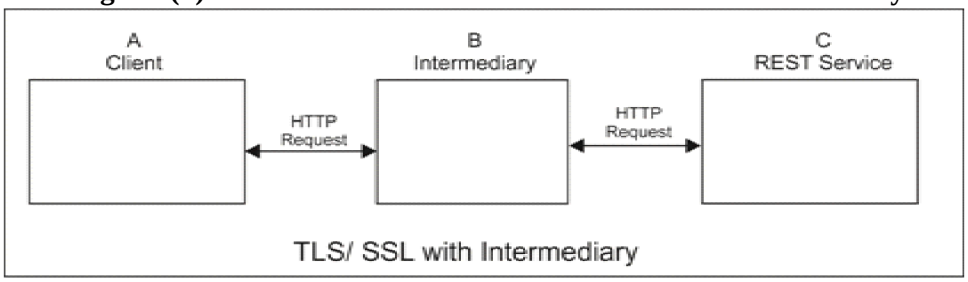
TLS/SSL in REST Web Services with Intermediary REST Relay Web Service

In this step, an actual REST web service (C), a REST client (A) and an intermediary REST relay web service (B) are created in the cloud environment. Both REST web service (C) and the REST clients (A) operate over standard HTTP -based REST protocols. The REST relay service (B) functions as an intermediary between the client and the actual REST service, and the communication binding used is web Http Relay Binding.

In this configuration, the intermediary relay web service (B) consumes the actual REST web service (C), while the client (A) consumes the relay service (B). Transport layer security (TLS/SSL) is configured at all three points – client (A), relay (B) and service (C). The HPSEA encryption and decryption algorithm is used for secure data processing during the experiment.

Using the relay web service as an intermediary allows for observation and analysis of sensitive data at various transmission stages using web debugging tools such as Fiddler or Wireshark. Since Transport -security provides protection only between two endpoints (e.g. from A to B or B to C), it becomes essential to inspect the encryption status of the data arriving at the intermediary (B). When encrypted data reaches the intermediary under TLS/SSL, iit is automatically decrypted by the receiving application at B. To maintain end-to-end confidentiality, the intermediary must re-encrypt and transmit the data over a new secure channel to the final destination (C).

Figure (6) - TLS/SSL at REST web service with Intermediary



(Source: Designed by Researcher)

Experiental Work

Implementing message-level security in REST web services enables end-to-end protection of data, as opposed to point-to-point protection offered by transport layer security (TLS/SSL). In this experiment setup, both web service and clients were tested using a text-based dataset.

In the experimental work, the researcher used text data set for measuring and recording performance of the implemented message security in REST web service including newly designed more secure and high performance encryption and decryption algorithm called HPSEA.

Table (1) Dataset used for performance evaluation

Name	Description	Data Size	Number of queries tested			
			GET	POST	PUT	DELETE
Text	Plain text	1KB – 16KB	10	10	10	10

(Source: Compiled by researcher)

In GET, POST, PUT or DELETE requests, data is signed and then encrypted before sending it through the POST request message body. Signed values of data are asymmetrically encrypted and symmetric key sent through custom headers. In HTTP, message reaches REST service, it is first decrypted using custom header data and then verified by applying encryption/decryption algorithm at the REST web service extensibility points. The messages processed by REST web service by creating a new resource on the REST service database and signed and encrypted response sent back to the REST client. Upon receiving signed and encrypted response at the REST client, it is decrypted and verified and finally presented at the client side. Image/audio/video data is converted to Base64 binary or Base64 string before sending to the REST web service." Jivtode M.L. (2021)

Table (2) GET execution Request / Response using HPSA algorithm for database

File size	Request time in milliseconds					Response time in milliseconds				
	Client Side			Server Side		Server Side		Client Side		
	Plain	Sign	Encrypt	Verify	Decrypt	Sign	Encrypt	Plain	Verify	Decrypt
1KB	330	-	-	-	-	203	52	1232	90	52
2KB	380	-	-	-	-	255	75	1249	109	63
4KB	394	-	-	-	-	329	107	1278	114	77
8KB	445	-	-	-	-	439	225	1297	169	124
16KB	586	-	-	-	-	474	245	1311	187	160

(Source: Compiled by Researcher)

Researcher conducted processing performance tests of custom headers and message body processing of REST web service in cloud environment.

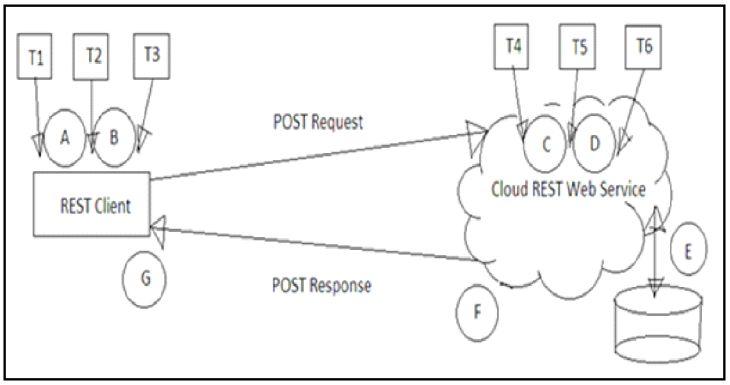


Figure (7) Request / Response timing calculation for POST

POST request/response timing are calculated by computing time elapsed between point A, B, C, D, E and F. POST request/response may result in retrieving resource from the local database on the client side.

Points A, B, C, D -> POST request

Point E -> Database processing

Points F, G -> POST response

Request time during GET = T2 – T1

Response time = Sign time + Encryption time + request time + Decryption time + Verification time.

POST Request/Response time calculation of extensibility point

i) Time T1 = DateTime.Now()

// Code for signing the data

Time T2 = DateTime.Now()

Point D -> Signing timing = T2 – T1

ii) Time T2 = DateTime.Now()

// Code for encrypting the data

Time T3 = DateTime.Now()

Point E -> Encrypting timing = T3 – T2

iii) Time T4 = DateTime.Now()

// Code for decrypting the data
Time T5 = DateTime.Now()
Point F -> Decrypting timing = T5 – T4

iv) Time T5 = DateTime.Now()
// Code for verifying the data
Time T6 = DateTime.Now()
Point G -> Verifying timing = T6 – T5

In the experiment, the researcher tested custom headers configured for use with REST web service. Headers are sent from REST client to service, carrying metadata like signing value, asymmetrically encrypted symmetric key etc. Picture shows message request/response to HTTP message headers. Here's HPSEA algorithm to measure the request and response time between a client and server using an Intermediary REST Relay Web Service. This algorithm considers three key timestamps -

1) T1 (Client Request Time): When the client sends the request.
2) T2 (Relay Receives Request): When the intermediary web service receives the request.
3) T3 (Relay Forwards Request): When the intermediary forwards the request to the server.
4) T4 (Server Receives Request): When the server receives the request.
5) T5 (Server Response Time): When the server processes and sends the response.
6) T6 (Relay Receives Response): When the intermediary web service receives the response from the server.
7) T7 (Relay Forwards Response): When the intermediary forwards the response to the client.
8) T8 (Client Receives Response): When the client receives the response.

1. Client starts timer: T1 = Current Timestamp
2. Client sends request to Intermediary REST Relay Web Service
3. Intermediary Relay Web Service records: T2 = Current Timestamp
4. Intermediary forwards request to Server: T3 = Current Timestamp

5. Server receives request: T4 = Current Timestamp
6. Server processes request and generates response
7. Server sends response: T5 = Current Timestamp
8. Intermediary receives response: T6 = Current Timestamp
9. Intermediary forwards response to Client: T7 = Current Timestamp
10. Client receives response: T8 = Current Timestamp
11. Calculate time delays: - Client to Relay: T2 - T1 -Relay to Server: T4 - T3 - Server Processing Time: T5 - T4 - Server to Relay: T6 - T5 - Relay to Client: T8 - T7 - Total Round Trip Time: T8 - T1

It has been implemented in Python

Client Script (client.py)

This script sends a request to the relay web service and measures response times.

import requests

import time

URL of the intermediary relay web service

RELAY_URL = “http://localhost:5001/relay”

Start time (T1) - Client sends request

T1=time.time()response=requests.get (RELAY_URL)

End time (T8) - Client receives response

T8 = time.time()

Display response and total round-trip time

print(“Response from server:”, response.text)

print(“Total Round Trip Time (RTT): {T8 - T1:.6f} seconds”)

Intermediary Relay Web Service (relay_service.py)

Forwarding the request to the actual server -

flask import Flask, request import requests

import time app = Flask(__name__)

URL of the actual web service

SERVER_URL = “http://localhost:5002/server” @app.route(‘/relay’, methods=[‘GET’])

def relay_request(): T2 = time.time()

Relay receives request response = requests.get(SERVER_URL)

Relay forwards to server T7 = time.time()

Relay forwards response to client return response.text

Return server's response if __name__ == '__main__': app.run(port=5001)

Run relay service on port 5001

Web Server (server.py)

from flask import Flask

import time app = Flask(__name__)

@app.route('/server', methods=['GET']) def server_response(): T4 = time.time()

Server receives request time.sleep(0.5)

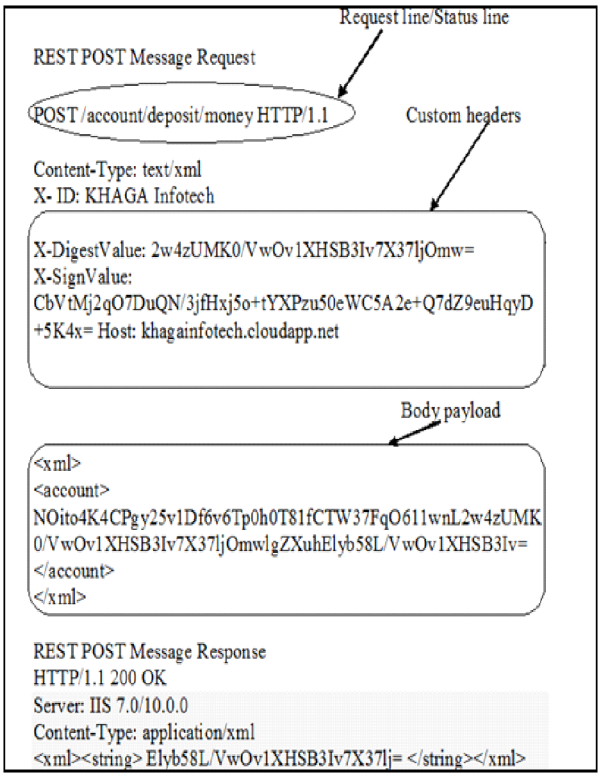
Simulate processing delay T5 = time.time() # Server sends response return f"Processed in {T5 - T4:.6f} seconds" if __name__ == '__main__': app.run(port=5002)

Run server on port 5002

Expected Output (Client)

Response from server: Processed in 0.500123 seconds

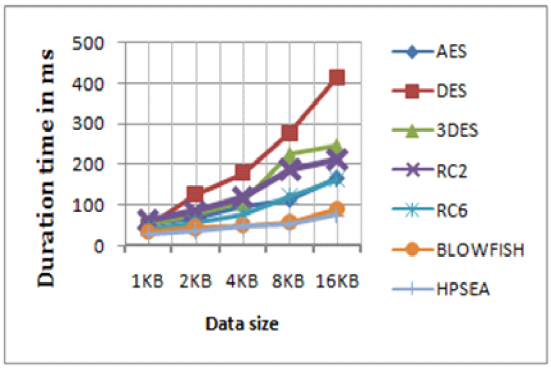
Total Round Trip Time (RTT): 0.602345 seconds



(Source: Compiled by Researcher)

The experimental results show the comparison of selected encryption algorithm and our HPSEA algorithm for plane text data and sizes, encryption/decryption speed, request/ response time of methods like GET, POST, PUT and DELETE.

Graph 1 encryption time (in ms) of encryption algorithm for plain text



Graph 1 illustrates the performance measurement of encryption/decryption of various data sizes versus time for each algorithm in REST message security services in the cloud environment for different operation using different algorithms.

It is clearly indicate that as the data size increases from 1KB to 16KB, the encryption and decryption time also increases. The analysis of encryption/decryption of HPSEA performs better compared to others algorithms in terms of the request/response time. Thus, it indicates that data size is directly proportional to encryption/decryption time.

Conclusion

This performance evaluation study of transport-level and message-level plain text data processing using intermediary relay web services provides valuable insights into their performance and reliability in distributed environments. The findings indicate that system performance is highly affected by factors such as network conditions, data size, processing overhead at transport and message layers, and the structure design of the relay web service.

The study also identifies that transport-level processing tends to offer better performance under low-overhead conditions, whereas message-level processing provides more

flexibility. Optimization techniques, like data compression, caching, and load balancing, as effective methods for improving both transport and message-level performance. Finally, intermediary relay web services play an essential role in managing plain text data in today's distributed systems. Further research could examine the application of dynamic load balancing and real-time analytics to optimize system performance in terms of scalability and reliability.

References:

1) Jivtode, M. L. (2021). *CRUD service for client and server timing computation using HTTP methods*. *International Journal of Engineering Research & Technology (IJERT)*, *10*(7), Paper ID:IJERTV10IS070022. https://doi.org/10.17577/IJERTV10IS070022
2) Jivtode, M. (2020). Message security in REST web services with intermediary REST relay service. *International Research Journal of Science & Engineering, Special Issue A7,* 784–790."
3) Aljawarneh, S., & Yassein, M. B. (2016). A security approach for web services against XML denial of service attacks. *Future Generation Computer Systems, 62*, 107–121. https://doi.org/10.1016/j.future.2016.02.007
4) Fielding, R. T., & Taylor, R. N. (2000). Architectural styles and the design of network-based software architectures (Doctoral dissertation, University of California, Irvine). https://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf
5) Puthal, D., Sahoo, B. P., Mishra, S., & Swain, S. (2015). Cloud computing features, issues, and security analysis. *Procedia Computer Science, 50*,33–42. https://doi.org/10.1016/j.procs.2015.04.184
6) Singh, R., & Pandey, S. (2019). Comparative performance evaluation of SOAP and REST web services using caching mechanism. *Journal of King Saud University – Computer and Information Sciences, 31(1)*, 84–92. https://doi.org/10.1016/j.jksuci.2017.06.004
7) Wang, Y., Shen, X., & Ma, X. (2020). Performance evaluation of RESTful web services under different load conditions. *Journal of Systems and Software, 165*, 110569. https://doi.org/10.1016/j.jss.2020.110569

8) J. Smith and A. Brown, "Performance Analysis of Web Services in Data Processing," *International Journal of Computer Science*, vol. 15, no. 3, pp. 45-58, 2022.

9) M. Zhang, L. Kumar, and P. Wong, "Transport-Level Security in Web Services: A Comparative Study," in *Proceedings of the IEEE International Conference on Cloud Computing*, 2021, pp. 234-240.

10) W. Davis, *Web Services Architecture and Performance Optimization*, 2nd ed. New York: Springer, 2020.

11) K. Patel and S. Mehta, "A Study on Message-Level Data Security and Processing Performance in Web Services," *Journal of Computer Networks and Communications*, vol. 18, no. 2, pp. 112-125, 2023.

12) C. Lee and H. Kim, "Relay Web Services and Their Impact on Data Processing Speed," *IEEE Transactions on Services Computing*, vol. 12, no. 5, pp. 678-690, 2019.

13) World Wide Web Consortium (W3C), "Web Services Architecture," Available: https://www.w3.org/TR/ws-arch/, Accessed: March 2025.

14) A. Gupta, R. Sharma, and P. Verma, "Comparative Analysis of XML and JSON in Web Services," *International Conference on Data Science and Applications (ICDSA)*, 2022, pp. 89-96.