Mesopotamian journal of Big Data
Vol. **(2025)**, 2025, **pp**. 329–349
DOI: https://doi.org/10.58496/MJBD/2025/021 , ISSN: 2958-6453
https://mesopotamian.press/journals/index.php/BigData

Research Article

Uniform Resource Locator Protection Scheme for the Mitigation of Man-In-The-Middle Stripping Attacks

Duaa Sameer Zhraw[1,*], Mohammed Abdulridha Hussain[1], Zaid Ameen Abduljabbar [1,2,3], Vincent Omollo Nyangaresi [4,5], Ali Hasan Ali [6,7,8], Husam A. Neamah [9,10]

[1] *Department of Computer Science, College of Education for Pure Sciences, University of Basrah, Basrah, 61004, Iraq*
[2] *Department of Business Management, Al-Imam University College, Balad 34011, Iraq*
[3] *Shenzhen Institute, Huazhong University of Science and Technology, Shenzhen 518000, China*
[4] *Department of Computer Science and Software Engineering, Jaramogi Oginga Odinga University of Science and Technology, Bondo 40601, Kenya*
[5] *Department of Applied Electronics, Saveetha School of Engineering, SIMATS, Chennai, Tamilnadu, 602105, India*
[6] *Department of Mathematics, College of Education for Pure Sciences, University of Basrah, 61004 Basrah, Iraq*
[7] *Technical Engineering College, Al-Ayen University, Thi-Qar 64001, Iraq*
[8] *Institute of Mathematics, University of Debrecen, Pf. 400, H-4002, Debrecen, Hungary*
[9] *Department of Electrical Engineering and Mechatronics, Faculty of Engineering, University of Debrecen, Debrecen, 4028, Hungary*
[10] *College of Engineering, National University of Science and Technology, Dhi Qar 64001, Iraq*

ARTICLEINFO

Article History
Received 21 Aug 2025
Revised 19 Sep 2025
Accepted 10 Oct 2025
Published 28 Oct 2025

Keywords
SSL Strip
MITM attack
HTTPS

ABSTRACT

Man-in-the-Middle (MITM) attacks reduce Hypertext Transfer Protocol Secure (HTTPS) to Hypertext Transfer Protocol (HTTP), compromising network communications to potential exploitation. Attackers exploit application-layer vulnerabilities, and the attack often occurs on LAN. This study addresses the problem by introducing a Uniform Resource Locator (URL) protection mechanism that combines encryption with secure key exchange.

A browser built with Python and PyQt5 encrypts URLs before transmission. The router decrypts, processes, re-encrypts, and returns data securely. The Diffie–Hellman algorithm generates a new session key for each connection, and the Advanced Encryption Standard with Galois Counter Mode (AES-GCM) technique to encrypt.

The system was tested in a VMware host-only environment under four scenarios: normal use, active attacker, system-only, and active attacker with the system enabled. Packet capture and timing analysis evaluated security and performance. The scheme achieved a 100% prevention rate against HTTPS downgrades. Intercepted traffic appeared as unreadable ciphertext. Average execution time increased from 0.05 seconds to 0.11 seconds due to encryption, but it did not affect stability.

This research improves application-layer security independently and offers a concrete defense against MITM stripping attacks. In conclusion, the proposed methodology provides a pragmatic and effective strategy for protecting URL traffic in vulnerable local network environments.

1. INTRODUCTION

The Internet has become a fundamental component of contemporary existence, significantly influencing several domains such as business, education, and employment [1]. It has transformed the realm of communications, reducing distances and streamlining tasks that once demanded considerable time and effort. One of its most significant applications is online banking, which has transformed the manner in which individuals and institutions conduct their financial operations [2]. In the digital era, nearly all transactions, whether direct or indirect, occur online. The high dependence on online services has necessitated the safeguarding of transmitted data to maintain its confidentiality, integrity, and accuracy[3].

The increasing reliance on digital platforms highlights critical issues, including the need for strong and flexible cybersecurity strategies to address rising threats. cybersecurity has become a key concern, primarily due to the extensive use of financial

*Corresponding author. Email: pgs.douaa.sameer@uobasrah.edu.iq

platforms, social media, and e-commerce systems that manage massive amounts of sensitive personal data [4]. In addition, the rapid technological advancements have raised risks, particularly the emergence of cyberattacks [5]. The recent past has witnessed evolutions in the cyberattack techniques, frequency and impact. This has immensely affected individuals, startups, and huge corporations. Data from existing studies reveal a consistent rise in attacks, with a significant surge in 2014, commonly designated as the "year of cyberattacks." Without effective and consistent technical solutions, these dangers are anticipated to persist and escalate [6]. Each year, attacks are increasing, mainly targeting major organizations, impacting information security, business continuity, and customer confidence. This escalating trend reached unprecedented levels in 2014, and in the absence of global solutions, this trend is likely to continue [7].

The proposed approach addresses this deficiency by consolidating various protective layers into a cohesive structure. Unlike prior works that concentrate on either detection or encryption alone, this system combines a secure key exchange mechanism based on Diffie-Hellman, AES-GCM encryption for all network communications, and a custom browser that ensures URLs and data are transported securely. Modular design principles further isolate encryption and decryption operations, enhancing maintainability and resilience against compromise. As explained in [8], MITM attacks are among the most serious security risks, which are not only prevalent but also detrimental. These attacks entail clandestine interception of communications between two parties, with the capacity to modify or seize the communicated data. MITM attacks are typically classified into two categories: MITM sniffing, and MITM stripping.

Due to their rapidly emerging significance within modern cyber threats, the purpose of this paper is to analyze MITM Stripping attacks. In these attacks, adversaries take advantage of vulnerabilities in security systems, which allow them to convert encrypted communications to unencrypted communications [9]. This allows leakage of sensitive data, including login credentials and financial information. A troubling aspect of MITM stripping attacks is that the victims often do not even know that their communication (which seems secure from their perspective) is compromised [10]. As such, this study makes the following contributions:

- We present a methodology for protecting against MITM stripping attacks by the application of encryption techniques.
- We utilize the Diffie-Hellman algorithm to facilitate safe key exchange between the client and the router. In addition, the AES-GCM algorithm is implemented for the encryption and decryption of network communications.
- A virtual environment that replicates the attack scenario is created, comprising a client device (Windows), an attacker (Kali Linux), and a router (OpenWRT).
- We develop a bespoke browser that encrypts URLs and transports them securely.
- We isolate the encryption and decryption procedures into distinct files in alignment with a security-oriented design.

The study is aimed at developing, deploying, and systematically testing a system that thwarts MITM stripping attacks by maintaining secure channels, encrypting messages, and exchanging keys safely. The value in this work is both practically and theoretically motivated. The system aims to create a new reference model that leverages high levels of encryption, secure key exchanges, and the modularity allowed by the proposed framework, thus providing an enhancement to protective measures against future advancements in cybersecurity of real-world networks, while also providing a reference model for future research and advancements in the study of both attack prevention, as well as secure communication.

The paper is structured as follows: Section 2 addresses the design of the virtual testbed environment, the client, attacker, router nodes, and the Host-Only network configuration. Section 3 covers the Python-based encryption system in a custom browser, together with the implementation strategies for all four test scenarios. On the other hand, Section 4 presents the proposed technique, which comprises end-to-end encryption using AES-GCM and secure key exchange using the Diffie-Hellman algorithm. Section 5 outlines the implementation specifics, encompassing the network architecture, operational protocols, and system setup. On the other hand, section 6 outlines the trial results across four specific scenarios and assesses performance metrics, such as reaction time and traffic attributes. Section 7 delineates the security validation of the proposed system. Section 8 closes this report and delineates prospective directions for further research. Section 9 delineates the limitations.

2. PRIMITIVE TOOLS AND ALGORITHM

2.1 Diffie-Hellman algorithm

The Diffie-Hellman key exchange protocol is one of the first methods used in public-key cryptography. It allows two parties to use an insecure connection to exchange the agreed-upon key in a secure manner. It involves two parties selecting a large prime number (p) and a base (g) for their Diffie-Hellman session. Each party selects their private key, then generates the corresponding public key from this private key. Thereafter, each entity shared their public key with the other entity. This public key is deployed to compute the shared secret key (based on entity's private key and the received public key). This shared secret key is treated as an anonymous shared secret and hence cannot be intercepted. The Secure Socket Layer (SSL)/Transport Layer Security (TLS) and Secure Shell (SSH) are two main protocols that utilize this key exchange process as part of their protocol specifications. This helps create a more secure connection for data transmission in a possibly adverse environment [11].

Numerous studies have examined the Diffie-Hellman algorithm's mathematical principles and practical applications, highlighting its advantages and disadvantages. Its major problem is its susceptibility to MITM attacks, in which an attacker can intercept and change public keys exchanged between users. To address this problem, previous researches have proposed combining Diffie-Hellman with other cryptographic techniques, such as digital signatures or RSA encryption. Test results indicate reduced execution time (0.0056 seconds compared to 0.0088 seconds for the classic Diffie-Hellman algorithm) and enhanced resistance to active attack [12]. The security of an algorithm depends primarily on the mathematical complexity of the discrete logarithm problem (DLP) within

cyclic groups. Therefore, the selection of significant prime numbers and appropriate generators is essential to maintaining cryptographic integrity. Several studies have investigated the relationship between the complexity of solving the Diffie-Hellman problem (DHP) and DLP, suggesting reduction-based methodologies for assessing the security of systems based on both problems [13].

This study uses the Diffie-Hellman method due to its logical framework, strong theoretical foundations, and importance in modern cryptographic systems. In addition, its integration with diverse encryption algorithms enables robust key exchange processes, which resists eavesdropping and manipulation threats. This renders it suitable for securing communication in untrusted and dynamic environments [14].

Note: The Diffie-Hellman method was used to make a dynamic session key between the client and router without the need for key management ahead of time. This solved the key management problem. This method makes sure that a new key is produced for each communication session and keeps the previous sessions hidden. This makes it less likely that someone will be able to access or use them, even if they find the next key.

2.2 AES-GCM Algorithm

The AES-GCM encrypt algorithm was selected due to its numerous benefits. For instance, it provides robust security and complies with the internationally recognized AES standard. In addition, it offers the fastest speed and allows real-time processing[15], and has widespread use in network protocols, such as TLS and Internet Protocol Security (IPsec). Moreover, the Diffie-Hellman method creates a shared secret key between client and router. Another significant benefit is that it is directly usable in encryption so that during a network transfer [16], both encryption and authentication are achieved. It also deploys number used once (nonce) values that are used in the encryption algorithm to ensure uniqueness and non-repetition in each encryption process.

Note: AES-GCM was chosen because it is faster and uses less computing power than public-key approaches. It both encrypts and authenticates in one step and utilizes a nonce to make each message unique. This means that you can't use the same message twice and that hacking is tougher. This means that it gives you good security that works well.

2.3 Tools

2.3.1 VMware Workstation 17 Player

VMware was employed to create a virtual environment consisting of the following three principal systems:

- **Router:** We configured a virtual router based on OpenWRT, which receives and processes encrypted data between the client and the attacker.
- **Client/Victim: This is a** Windows-based virtual machine that represents the protected user. It is used for navigation and to transmit encrypted data**.**
- **Attack:** This is represented by a virtual machine running Kali Linux that performs MITM Stripping attacks to determine the effectiveness of the proposed solution.
- **VMware network configuration:** Two networking types were created to connect the virtual machines.
- **Host-only network:** Created a connection between the client and the attacker, as well as the connection to the router within a confined internal network. Static IP addresses were assigned to each device to ensure consistent communication.
- **Bridged network:** Configured on the router to enable connectivity to the external internet during practical testing scenarios.

2.3.2 Wireshark software

Wireshark is a network protocol analyzer that captures, examines, and analyses network traffic in real-time. It enables comprehensive packet-level analysis, making it possible to identify vulnerabilities, monitor network activity, and detect potential security threats.

2.3.3 Python and utilized libraries

All procedures related to key exchange, encryption, and decryption were executed using Python 3.11. The following libraries were employed: *socket* for enabling inter-device connectivity; *cryptography* for executing AES-GCM encryption; *pyDH* for performing Diffie-Hellman key exchange; and *PyQt5* to create a custom browser on the client device.

3. RELATED WORKS

In this section, we discuss some of the previous works dedicated for MITM mitigation. For instance, the research in [17] investigated SSL-based session hijacking attacks, with special focus on SSL stripping. The authors categorized current preventive mechanisms against SSL stripping attacks as client or server-side solutions. This classification highlights the strong dependence of existing defenses on lower-layer mechanisms, which limits their ability to secure data at the application level. The various components in their system included data proxies, static Address Resolution Protocol (ARP) tables, Extended Validation Secure Sockets Layer (EV SSL) certificates, two-way authentication, HTTPSLock, SSLock, HTTP Strict Transport Security (HSTS), ISAN-HTTPS port, ShS-HTTPS port, and cookie proxies. Among the tools and technologies deployed included browser caching systems, web script-based solutions, and ARP-based alternatives. Regarding performance, the several fixes showed different degrees of success. Strong security against vulnerabilities came from two-factor authentication and HSTS. Although their approach has various shortcomings

(such as sensitivity in initial user confidence and significant implementation costs), it provides a fundamental knowledge of the attack concept and its execution. In addition, these techniques show notable restrictions on browser compatibility, usability issues, and resource economy problems. Moreover, the authors noted that no best measure would help reduce all dangers while maintaining user-friendliness and cost-effectiveness. However, the study did not address how URL requests themselves can be encrypted or protected at the endpoint before entering the transport layer, which represents a clear gap that this research aims to fill.

This research done by Chordiya et al. [18] discusses the central topic of MITM, and specifically HTTPS session hijacking attacks enabled by SSL stripping and ARP spoofing undergraduate. The authors showed that it is possible to capture and modify secure communications between clients and servers using the ARP and SSL weaknesses. The study mainly focused on demonstrating the feasibility of the attack and did not extend its scope to evaluate defense strategies or protocol-level improvements. The target was Ubuntu, and Kali Linux was the attacking machine. The custom scripts were run on a virtual machine environment using Wireshark and Ettercap as tools. The method uses SSL stripping to change the HTTPS connection to a process referred to as HTTP cache poisoning. In addition, attackers could redirect traffic and capture sensitive data, such as unencrypted intermediate login credentials. Thus, the method highlighted the vulnerability of HTTPS connections to MITM attacks. However, this method has some specific limitations, such as requiring the attacker to be on the same local network as the victim. In addition, it cannot bypass HSTS security. Another limitation is that the evaluation focused only on the attack scenario without analyzing how different network configurations or endpoint-level protections could affect the success rate, which limits the generalizability of the findings. Nonetheless, this study highlights attack paths and helps with understanding HTTPS vulnerabilities. Nonetheless, it lacks a description of defensive methods for these types of attacks, which limits its general applicability and usability in both practical and academic cybersecurity contexts. The lack of defensive strategies identifies a definitive research gap that supports the proposed new strategies focused on endpoint security and encryption at the application layer.

The research by Jonas et al. [19] dealt with SSL stripping and addresses the crucial problem of session hijacking vulnerabilities. The study approached the issue from a usability and user-awareness perspective rather than introducing a fundamental change at the protocol or application layer. In addition, it offers an ideal solution that combines security with usability. Using a Naive Bayes classifier, the authors developed a system to assess websites, including multi-tiered warning messages based on on-site security relevance. They employed machine learning methodologies to assess user behavior and website assessments. This approach utilizes ongoing rating adjustments informed by user activity to assess a 50% regression threshold via split-half reliability testing and correlation analysis. The methodology incorporates machine learning systems, exploratory methods, and statistical analysis. Basically, it adapts through iterative learning from human input. While this behavioral approach provides value in user education, it does not address how attackers can exploit lower-layer weaknesses to bypass browser-based warnings, leaving a critical blind spot in actual network attack scenarios. Its primary benefits include enhanced security for critical websites, reduced superfluous notifications for less significant sites, and the ability to educate individuals on safe browsing practices incrementally. Nonetheless, it depends on unrefined datasets and is vulnerable to interference during execution. In addition, it has complexities during installation and can potentially post false positives and negatives during the learning period. Another limitation is the lack of evaluation against active stripping attacks in real networks, which restricts the ability to verify the robustness of the proposed solution beyond simulated user environments. This creates a clear gap for mechanisms that provide direct application-layer protection independent of user behavior. These setbacks may impact overall security effectiveness and user experience.

The study by Duddu et al. [20] addresses the primary concern of MITM attacks, particularly SSL stripping techniques that undermine HTTPS security. Their focus remained largely on reinforcing existing transport-layer defenses rather than introducing new protective measures at the endpoint or application layer. The authors outlined the approaches to protecting against SSL strip attacks while considering weaknesses related to self-signed SSL certificates. This research looked at multiple attack vectors, which included ARP spoofing, and suggested countermeasures such as HSTS. The results indicated that HSTS and trusted certificate authorities could significantly reduce the risk of MITM attacks. While effective in controlled environments, these measures depend heavily on strict browser support and correct server configurations, which creates deployment inconsistencies in real networks. The main benefits are protection of HTTPS connections and a better understanding of certificate validation. The paper not only offers valuable methods for stopping SSL strip attacks but also considers some possible ways to improve HTTPS security systems. However, challenges still exist with the widespread deployment of HSTS, which limit its practicality. In addition, the lack of technical details in its implementation strategy hampers replication. Another limitation is the absence of evaluation against adaptive attackers capable of exploiting endpoint behavior or mixed-content scenarios, which leaves gaps in understanding real-world attack resistance. In addition, practical significance was diminished as the research limited itself to LinkedIn as a platform. This limited its value and lacks clear, rational means of reducing attacks. This narrow scope and focus on transport mechanisms highlight the need for complementary application-layer protections, which this work aims to address.

The research by Adjei et al. [21] describes an underlying problem caused by ARP spoofing and Dynamic Host Configuration Protocol (DHCP) - based MITM attacks on LAN. Their work mainly focused on detecting and blocking network-layer attacks but did not propose mechanisms that operate at the application level, which leaves higher-layer data exposed if network defenses are bypassed. To effectively mitigate this problem, the authors presented a detailed prevention plan, which combines DHCP monitoring and ARP spoofing. Using technologies such as GNS3 and Wireshark, the authors implemented the attacks, detection, and mitigation processes with experimental simulations. They presented a flexible methodology that would work successfully on a variety of networks and mitigate SSL stripping among common local area attacks, such as the ping of death (PoD). Its limitations include potential conflicts with network configurations, as well as issues related to on-site implementation. The authors adopted a broad range of software tools through multiple operating systems to establish a complete experimental technique that included initiating,

detecting, and mitigating. While their approach may be methodical, it requires maintaining constant monitoring, and for full control of the network infrastructure, it might not be employable, for example, with resource-limited routers with a lack of administrative access. Also, doing so puts extensive demand on computational resources to monitor everything in real-time. However, this approach may require significant processing capacity for real-time monitoring. In addition, its efficiency may be affected by dynamic network conditions monitoring. In addition, its efficiency may be affected by dynamic network conditions. One shortfall is that there is no analysis on how the defenses would behave under adaptive attackers and/or encrypted traffic, which limits the defenses' applicability outside of the controlled lab settings. Nevertheless, this study provided the foundational understanding necessary to develop effective security plans for mitigating SSL stripping and related MITM attacks in local area networks. This focus on network-layer mitigation highlights the absence of complementary application-layer protection, a gap that this research seeks to address.

Kampourakis et al. [22] investigated MITM attacks on HTTPS by narrowing down the problem to gaps in behavior, configuration, and advanced circumvention techniques that defy HSTS security. This approach focused on detecting vulnerabilities at the browser and configuration level without proposing corresponding protection mechanisms, leaving practical defensive strategies unexplored. The authors advanced the traditional MITM attacks by carrying out some real-life demonstrations, with the intention of providing approachable visualizations for new and alternative attack methods. This included developing reasonable attack vectors, attacking the current browsers and systems, and identifying browser vulnerabilities such as HSTS variations, embedded HSTS exploits, and other weaknesses based on particular browsers. They showcased the gaps that exist in the security measures put in place within the HTTPS system, and the loose user engagement with HTTPS implementations. Some noted limitations include the specific assumed mid-user configured versions of the misaligned browser, as well as the mid-user versions that do not represent many real-world scenarios. These assumptions reduce the validity of the results externally and limit their applicability to broader network environments. The major contributions of this study included the aggressive examination of contemporary MITM and man-in-the-browser attack strategies, as well as the lack of supporting research to address vulnerabilities in HTTPS implementations. The authors point out that despite moving towards closing security gaps in HTTPS, it is troubling that that there are no approaches to proactively mitigate vulnerabilities. This lack of proactive protection strategies highlights a gap in application-layer encryption mechanisms that can work alongside existing browser protection mechanisms, which this research seeks to address.

To address an important issue of MITM attacks on mobile applications, Ali et al. [23] analyzed the limited abilities of bad SSL/TLS implementations on Android and iOS systems. The study focused on highlighting vulnerabilities in mobile application communications without providing protection mechanisms or a systematic assessment of potential defenses, leaving critical gaps unaddressed. The authors investigate how attackers take advantage of mobile applications to collect private user data passing through communication channels. To demonstrate the effectiveness of these attacks, they deployed benign software on non-rooted devices, using Wireshark and proxy servers as traffic interceptors to alter data flows. They identified critical vulnerabilities in the security of mobile application interfaces, which were a significant focus of their analysis. This methodology has shown serious flaws but remains descriptive, providing no structured methodology for mitigating them or integrating with secure application layer frameworks. However, their lack of definitive solutions or countermeasure frameworks was the major drawback of their work. Nevertheless, the work aimed to raise awareness of SSL and TLS vulnerabilities in mobile applications, and paved the way for further studies whose goal is directed towards proactive and preventative research. The study also ignored dynamic attacker behaviors, such as adaptive MITM malware or scenarios involving compromised local networks, which limits the practical applicability of the results. The restrictions of this study include the dearth of sophisticated solutions, lack of attention to malware, or rooted devices apps, as well as the absence of comparative studies on mobile devices that would be very helpful in advancing knowledge on mobile devices. In addition, this research also assumes that mobile devices do not have MITM malware, further compounding its limitations. Such assumptions reduce the external validity of the study and highlight the lack of a robust, deployable protection mechanism at the application layer for mobile platforms. The authors point out some inconsistency where hardware architectures are not sufficiently secured due to the assumption made during the construction of mobile apps, regarding the effective security of SSL and TLS.

Fereidouni et al [27] examines flaws that could be exploited in multiple levels of the Internet of Things (IoT) architecture. This work provides a broad overview of vulnerabilities but does not provide a concrete defensive framework, especially at the application layer where endpoint protection remains limited. In addition, the authors studied the limitations faced by resource-constrained devices in the context of threats posed by MITM attacks in the Internet of Systems. The study suggested that a comprehensive review of the existing attack vectors and alternative defense measures be conducted. In addition, other techniques such as rate limiting, timers, and machine learning based detection systems need to be deployed to create checks for the MITM defense. These proposals remain conceptual and are not accompanied by practical deployment models or empirical evaluations in heterogeneous IoT environments. Their methodology included examining traditional machine learning models such as logistic regression, random forests, and decision trees. In addition, state-of-the-art techniques such as genetic algorithms and bidirectional long short-term memory (BiLSTM) combined with dimensionality reduction via principal component analysis (PCA) were studied. The findings indicate that classical models can achieve 99% accurate results on simpler datasets. However, this high accuracy is achieved under controlled conditions and does not reflect the operational diversity or resource constraints typically found in IoT networks, calling into question the generalizability of the results. However, the robustness and complexities presented by IoT networks and the diversity across these networks limit its practical applications. In addition, some barriers do exist, such as the absence of uniform IoT security terminology, dataset accessibility, and the tradeoff of security mechanisms and the performance of the device itself. The value of this research includes the emphasis on new technologies such as deep learning, which will continue to improve the detection ability, and the proper analysis of IoT MITM vulnerabilities. The study demonstrates that more research needs to be done on scalability and

adaptability of networks in the IoT's dynamic scenarios. As such, there is no conclusive way to address the gaps for all situations. This highlights the absence of a unified, lightweight, and deployable security mechanism that can operate at the application layer of IoT systems. This study enhances the understanding of the MITM vulnerabilities, as well as the countermeasures for IoT security threats. It produces meaningful insights that contribute to the improvement of security and resilience of IoT mechanisms.

Previous studies on the prevention of MITM attacks have primarily focused on prevention at both the transport and network levels, especially SSL stripping and ARP spoofing. HSTS enforcement, two-factor authentication, certificate validation, network traffic monitoring, and user behavior alerts are examples of the methods that researchers have experimented with to slow attacks down or to protect the user from their exposure. Typically, researchers have used some combination of existing tools (Wireshark, Ettercap, machine learning classifiers, etc.) to identify shortcomings in each of these techniques. While these steps have made users more aware of identifying attack vectors, they each have their own limitations. Most solutions depend a lot on security at lower layers, are sensitive to how well browsers function together, need a lot of money to install, or presume that networks are controlled. Most significantly, they don't protect application-level URL request encryption, which makes endpoints easy targets for direct attacks. The proposed approach fills this gap by adding URL-level encryption before transmission. This provides strong application-level protection that works regardless of network conditions or user behavior. This method stops SSL stripping attacks at the source, makes endpoint security better, and works with existing transport layer protocols. It is a practical and deployable solution that can protect against real-world MITM threats on a wide range of systems and devices.

Unfortunately, there is a lack of empirical data or practical analysis from the identified solutions. In addition, these studies assume that all devices have the capability of implementing complex solutions, notwithstanding their limitations. Table 1 presents a summary of some of the outstanding works in this domain.

TABLE I: SUMMARY OF RELATED WORKS

Source	Year	Method || Algorithm Used	Strengths	Weaknesses
Hossain et al.	2018	Date proxy, static ARP table, EV SSL certificate, two-way authentication, HTTPS Lock, SSLock, HSTS, ISAN-HTTPS port, SHS-HTTPS port, and cookie proxy.	Explains the strengths and weaknesses of each security method, as well as practical steps for implementing the attack using opensource tools such as SSL Strip.	Lack of a comprehensive solution that effectively mitigates all risks while maintaining efficiency and ease of use.
Chordiya et al.	2018	ARP spoofing, SSL stripping, Ettercap, SSL strip	Illustrate attack paths and raise awareness of HTTPS vulnerabilities.	Insufficient focus has been placed on preventative solutions for such attacks, diminishing their practical and educational importance.
Jonas et al.	2019	Naive Bayes algorithm for keyword classification based on their content and URLs	Enhancing security for core websites, can educate consumers on safe browsing practices in a gradual manner.	Reliance on primary datasets and potential noise during early deployment stages.
Duddu et al.	2020	ARP poisoning with HSTS and HSTS, CA validation	Enhance security with HTTPS connections; increase vigilance regarding certificate verification.	The implementation method lacked technical details, making replication difficult, quantitative results demonstrating the effectiveness of the attack were absent.
Adjei et al.	2021	A solution developed using DHCP Snooping and Dynamic ARP Inspection.	An overall license that includes initiation, detection, and mitigation as part of a foundation that requires software that could be used over multiple operating systems.	The approach may require a lot of processing resources for real-time monitoring, and the effectiveness may also be reduced in very dynamic networking environments.
Kampourakis et al.	2022	Browser attack testing and scenario analysis	It's a comprehensive examination of contemporary MITM attack methodologies and focuses on uncovering overlooked vulnerabilities in HTTPS implementations.	It doesn't offer complete solutions, but works more to reveal and describe vulnerabilities.
Ali et al.	2022	Mobile app SSL/TLS analysis and traffic interception tools	Increases awareness of SSL/TLS vulnerabilities found in mobile applications, sets the stage for future research concerning taking measures for security protection.	Inadequate countermeasures, limited focus on scrutinizing benign applications.
Fereidouni et al.	2025	Multi-layer with deep learning algorithms such as Random Forest, Autoencoders, LSTM, and PCA.	Thorough analysis of IoT-specific MITM vulnerabilities; emphasis on upcoming technologies, such as deep learning.	Dataset access issues, scalability problems

4. THE PROPOSED SCHEME

This model tackles the MITM stripping attack problem through five discrete and sequential phases: *initialization*, *generation and key exchange*, *client-side URL encryption*, *data transmission to server*, and *response transmission to the client*. The penultimate stage involves the router sending data to the server, which then creates some response. The encrypted response is then sent to the client. To ensure connection integrity and prevent any attempts to intercept or modify the data, each phase is built upon the previous one, as demonstrated in Figure 1. Table 2 presents the symbols and notations used throughout this paper.

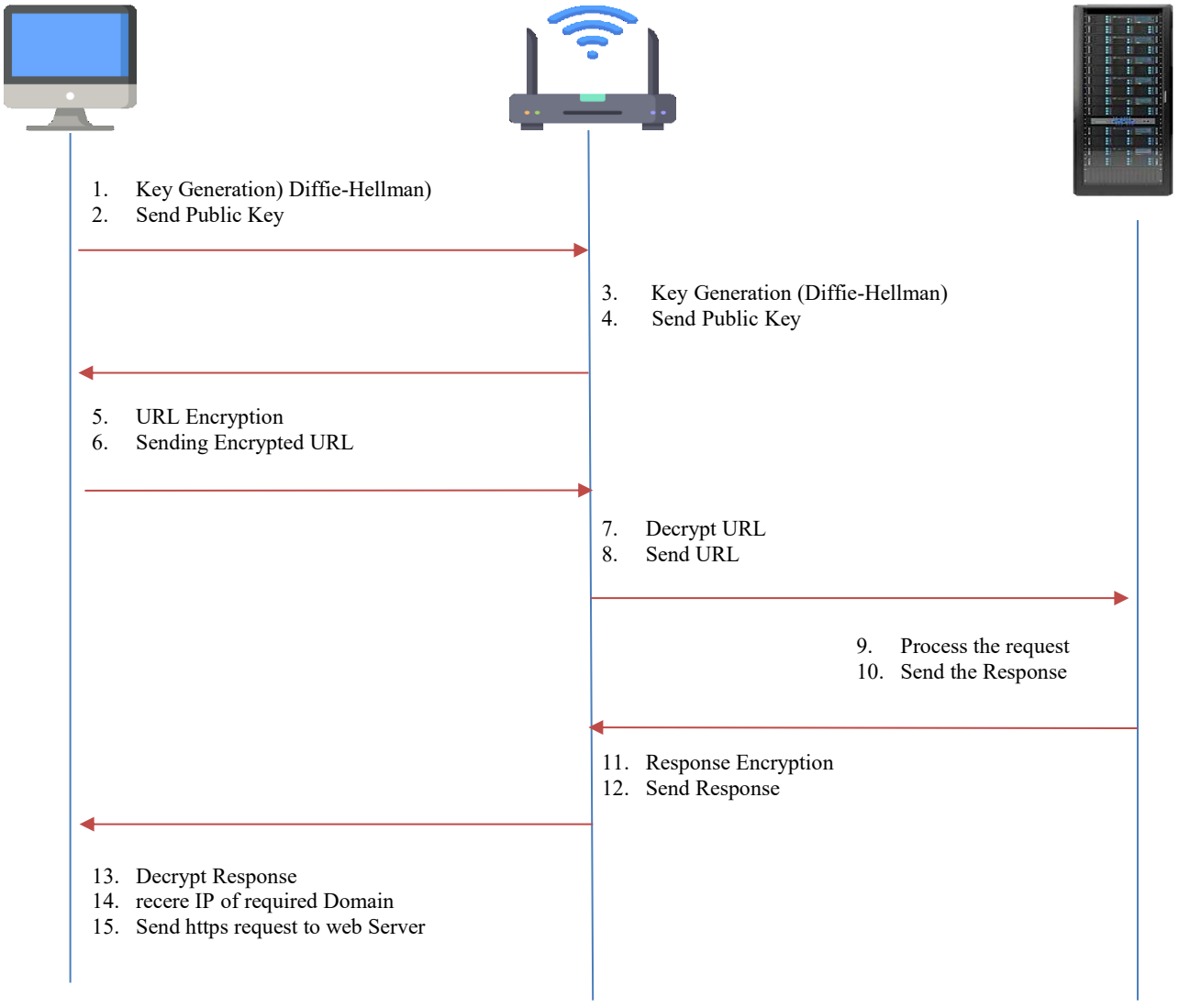

Figure 1: Structure of the proposed technique for the MITM stripping attack

TABLE II: SYMBOLS AND NOTATIONS

Notation	Description
g, p	Diffie Hellman initial parameters
C	Client
R	Router
S	DNS Server
WS	Web Server
E	AES-GCM encryption algorithm
D	AES-GCM Decryption algorithm
a, b	Private keys
A, B	Public Key
DQ	DNS Query which contains domain name
DR	DNS Reply which contains IP of the Web server
URL	Uniform Resource Locator which contains the protocol

QID	Query ID
HTTPS req	HTTPS Request contain URL
HTTPS resp	HTTPS Response contain HTML

Protocol 1: Execution summary

C:	Generate (a)
	Compute ($A = g^a \ mod \ p$)
$C \rightarrow R$:	A
R:	Generate (b)
	Compute ($B = g^b \ mod \ p$)
	Compute ($K = A^b \ mod \ p$)
$R \rightarrow C$:	B
C:	Compute ($K = B^a \ mod \ p$)
	Generate Nonce (n)
	$EU = E_{k,n}(URL, QID)$
	Browser Generate (DQ) message
$C \rightarrow R$:	DQ
R:	Generate Nonce (n)
	$D_{k,n}(EU)$
	Append *URL* to *DQ*
$R \rightarrow S$:	DQ
$S \rightarrow R$:	DR
R:	Generate Nonce (n)
	$EU = E_{k,n}(DR, QID)$
	Append *EU* to *DR*
$R \rightarrow C$:	DR
C:	$D_{k,n}(EU)$
	Obtain *IP*
	Generate *HTTPS req*
$C \rightarrow WS$:	*HTTPS req*
$WS \rightarrow C$:	*HTTPS resp*
C:	Browser Display website page

Where: nonce (n) = *HMAC* ($H(K)$+ index)

Phase 1: Initialization phase

This phase configures the primary connection between the client (Windows) and the router (OpenWRT). The main aim is to generate two identical values (P and G) in the Diffie-Hellman key exchange algorithm:

- **P:** a Large prime number representing the domain for the computations performed.
- **G:** an integer that is comparatively lesser than P, designated as the generator of a numerical set.

The numbers must be congruent before being communicated between the two parties, as any disparity will hinder the generation of a shared key. These numbers are either generated randomly at a specified moment on either side or carefully chosen. Upon execution, both parties become ready to proceed to the next phase. The initialization phase aims to provide a dependable mathematical framework that can later be employed for data encryption and decryption.

Phase 2: Generation and Keys Exchange

After the generation of the P and G values, the client and router perform the key generation step of the Diffie-Hellman algorithm. In this approach, each member generates a private key, exchange the related public values, and compute the shared key. The key is hashed (calculated using a one-way function that utilizes numbers that cannot be decoded with the available computer power at the current time). The final key is not necessarily communicated; both parties derive it separately. As demonstrated in Figure 2, this key is used to encrypt and decrypt all the communications. As such, it serves as the main assurance of session confidentiality between the connection and the router.

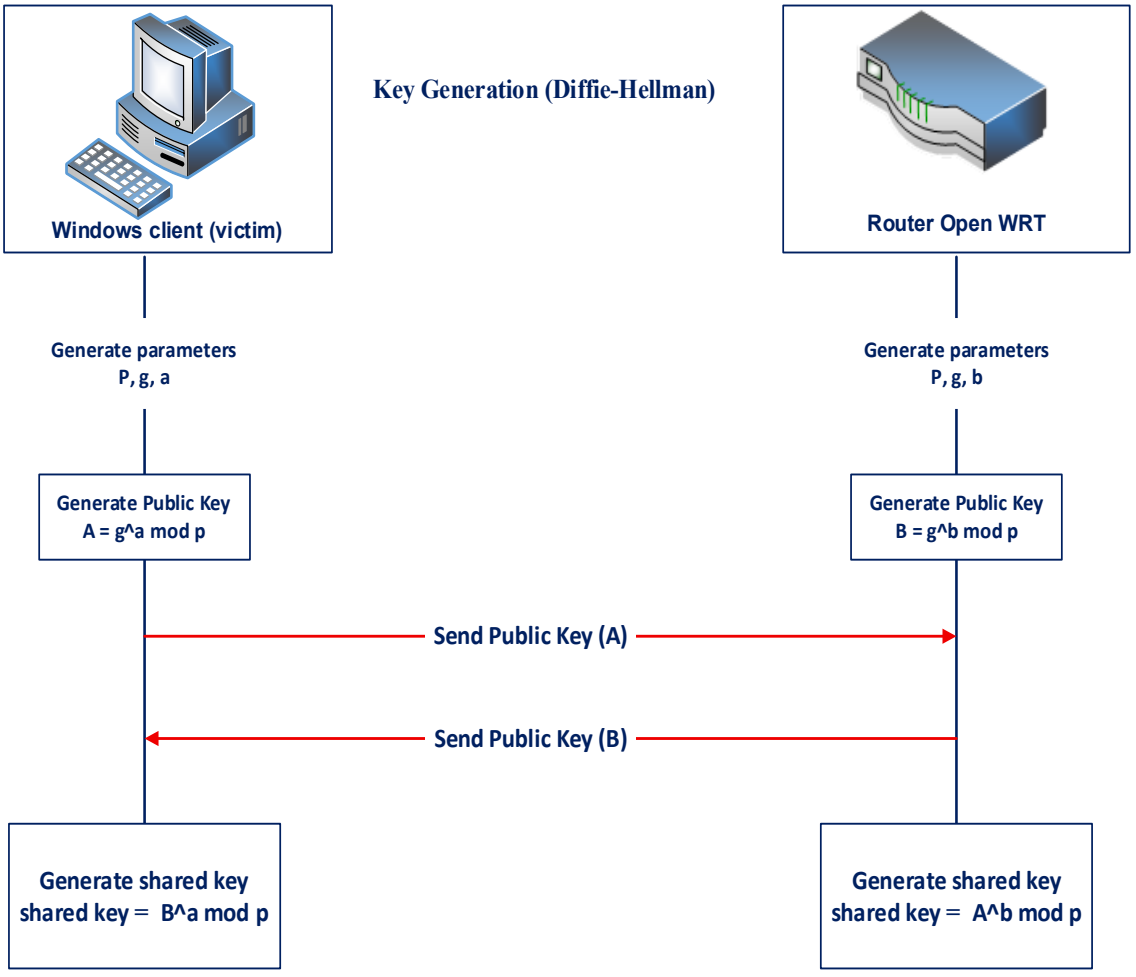

Figure 2: Key generation and exchange

Phase 3: Encryption and transmission of URLs

Encryption begins after a user submits a URL within the custom browser. Prior to encryption, the query is paired with a unique *query-id*, which is an authentication and sorting identifier on the router side. The AES-GCM algorithm is then applied to this data for encryption. The random token is not arbitrarily generated or exchanged across the network. Rather, it is derived deterministically by calculating the Hash-based Message Authentication Code (HMAC) of the shared session key using a fixed hash-based base-key. The final nonce is calculated as follows: The base-key is a cryptographic hash (for example, Secure Hash Algorithm 256-bit (SHA-256)) of the shared key generated from Diffie-Hellman. On the other hand, the index is an incremental integer counter for each query, starting at zero. The final encrypted data, which includes the query and *query-id*, is packaged in JSON format and sent over User Datagram Protocol (UDP) to the router using a similar method as in Figure 3.

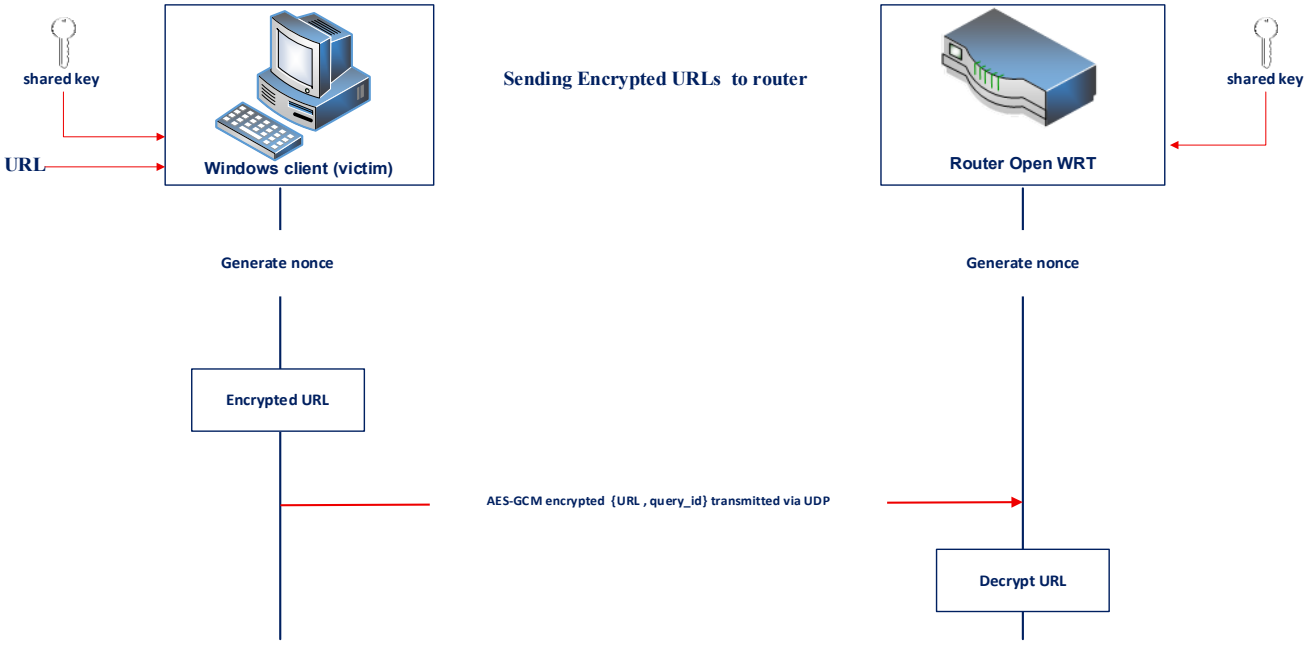

Figure 3: Sending encrypted URLs

For enhanced security, nonce values are never transmitted over the network. Both parties independently generate the same nonce by taking the shared session key (base-Key) and producing a hash with a mutually agreed-upon query index. This guarantees forward security and alleviates the risks associated with nonce reuse or interception.

Phase 4: Data transmission to DNS server

In this phase, the router receives the URLs, which then deploys the AES-GCM algorithm to decrypt these URLs. In addition, the router possesses the decryption key and nonce. Subsequently, the router obtains the request from the DNS server and transmits it to the client. Basically, the DNS server receives the request, processes it, and then sends the result to the router, as demonstrated in Figure 4.

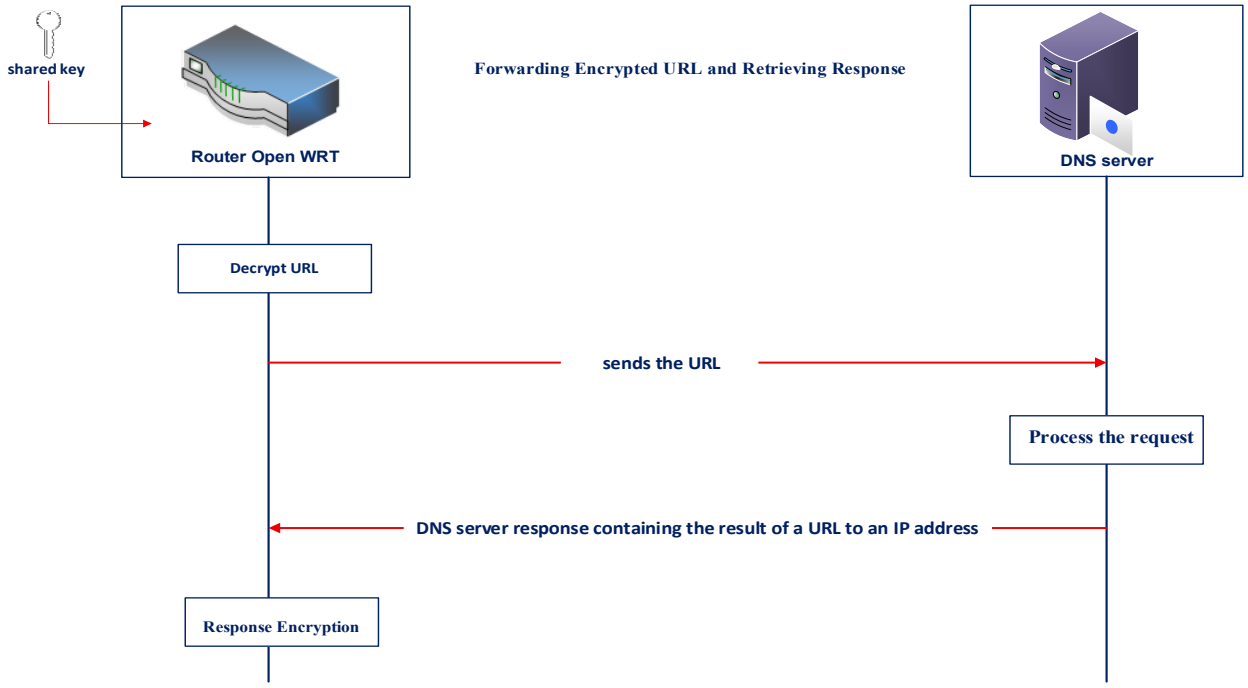

Figure 4: Data transmission to DNS server

Phase 5: Management of encrypted responses

In the final stage, upon getting the result from the DNS server, the router promptly encrypts it using the same shared key generated in the second stage. Next, the data is sent over a secure layer with AES-GCM encryption and a new nonce. This is followed by the transmission of the web request towards the client machine. Upon getting this request, the client retrieves the key from the *temp* file and deploys the nonce to decrypt the message. Once the client has deciphered the content retrieved from the DNS server, it forwards

an HTTP/HTTPS request to the web server. Basically, it uses the IP address to obtain the contents of the website, which is then displayed in the client's browser. After this, the connection comes to its end and the session is now considered fully active. Figure5 represents an active session where there has been no attempt by an intermediary to change or alter the contents of this session.

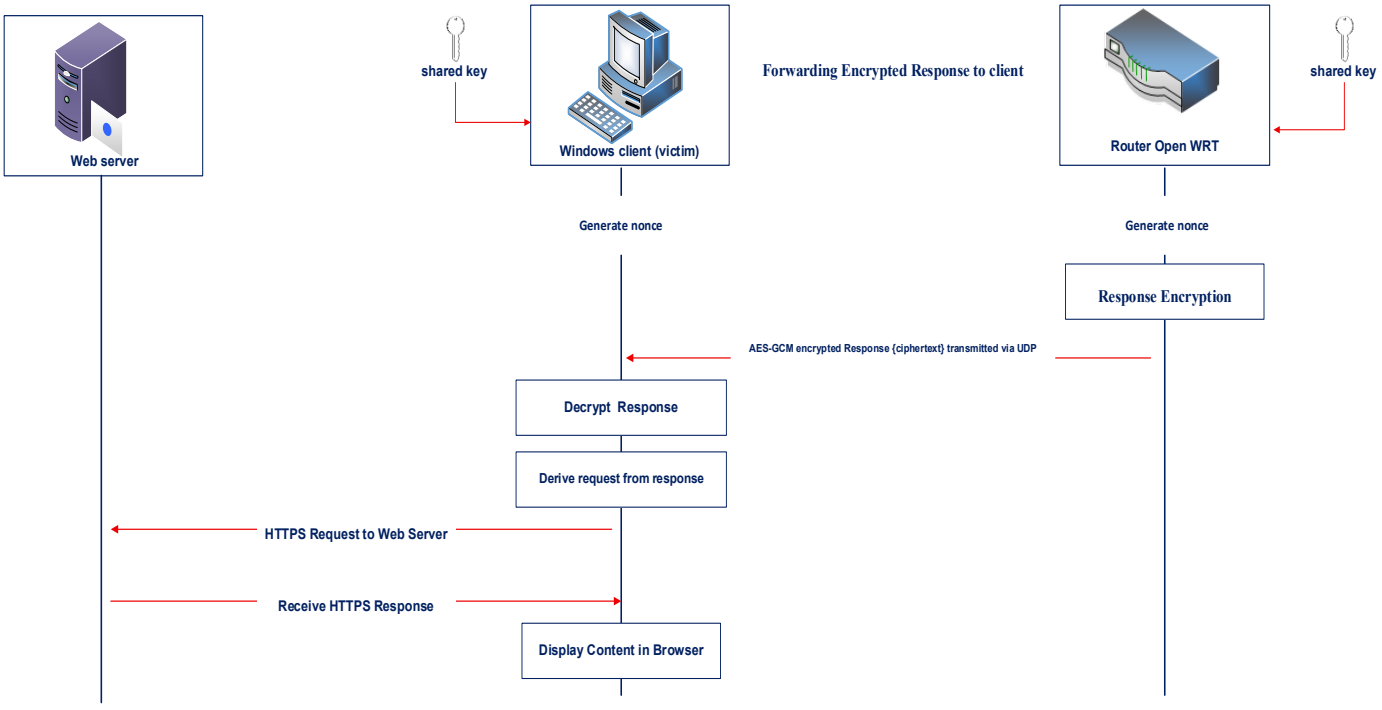

Figure5: Transmit HTTPS response to the client

5. IMPLEMENTATION

This section provides the detailed description of the network architecture, explains the practical implementation steps, explains the proposed solution to the MITM stripping attack problem, and analyzes its effectiveness.

5.1 Network Model/Architecture

In this work, a Virtual Local Area Network (VLAN) was built in a VMware Workstation 17 Player environment to simulate a MITM stripping attack and test the proposed solution under semi-realistic conditions. There are three core nodes in the proposed network model: the client, attacker, and the router. Here, the client is running a Windows 10 operating system, running a user conducting searches in a custom browser. On the other hand, an attacker is a Kali Linux system running MITM attack applications, including SSL Strip and Ettercap. On the flipside, the router comprises an OpenWRT system modified with Python scripts for Diffie-Hellman key exchange and AES-GCM message decryption.

The router gets internet connectivity from the physical computer via a bridged network. All devices can communicate with each other through the router's host-only network, isolating the entire environment from the outside internet. Therefore, the router serves as the source of these communications. Here, the client device runs a specially designed Python browser, using the PyQt5 framework.

Using a shared secret key generated using the Diffie-Hellman protocol, this browser encrypts URLs before sending them to the router via a specific port (1650). The network structure is illustrated in Figure 6.

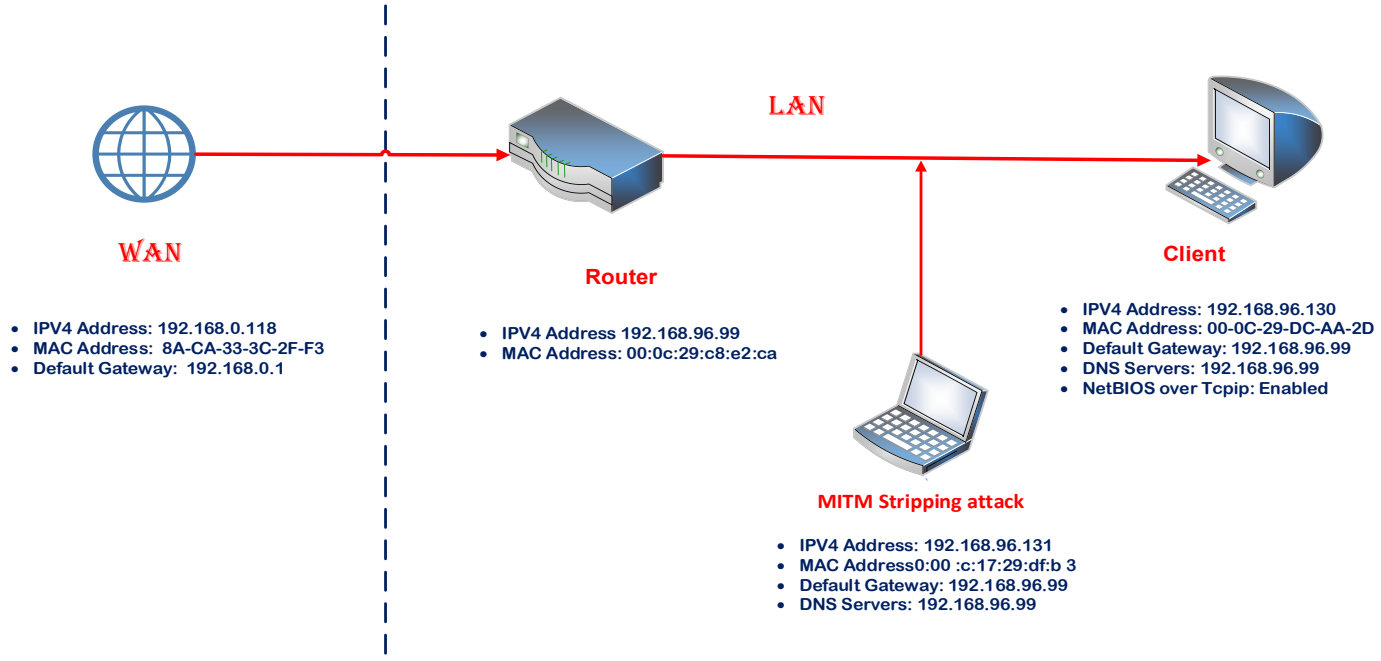

Figure 6: Network model

The router serves as the center node in the network, receiving all connections from client devices. To enable the attack employing protocol corruption methods (ARP spoofing) and in an effort to diminish the encryption level (SSL stripping) prior to and subsequent to the implementation of the proposed system, the attacker's equipment is situated on the same subnet.

5.2 Execution procedure

The proposed method was implemented in a carefully designed simulation environment that mimics a real LAN. This environment consists of a client computer (Windows), an attacker computer (Kali Linux), and a custom router running OpenWRT. This system aimed to evaluate the system's effectiveness against MITM stripping attacks across four phases: scenario1 (ideal environment), scenario 2 (ideal environment with an active attacker), scenario 3 (the proposed system environment), and scenario 4 (environment with an active attacker and the proposed technique).

The system is built using Python, with PyQt5 being used to create a custom web browser interface that allows URL entry. These URLs are encrypted using the AES-GCM algorithm, with both a shared key (pre-generated via Diffie-Hellman key exchange between the client and router) and a nonce. This ensures that the result of each encryption operation is different, even if the same link is repeated. The base key and nonce are generated as follows:

base_key = *hashlib.sha256*(*shared_key*).*digest*()

nonce = *HMAC* (*base_key*, *index*)

After the URL and *query-id* are encrypted, the data is encapsulated into a JSON structure containing a cipher-text as follows:

payload = *json.dumps* ({"*ciphertext*": *encrypted_ciphertext_hex*}).*encode* ()

This data is then converted to bytes and sent over UDP as follows:

sock.sendto(*payload*, (*ROUTER_IP*, *ROUTER_RECEIVE_PORT*))

Upon receiving the response from the router, the browser decrypts the data (URL) using the same key and new nonce, as follows:

response_dict = *decrypt_data* (*shared_key*, *nonce*, *ciphertext_response*)

The router's role in the proposed system, it receives encrypted data (URL) from the client via UDP and decrypts it using the shared key and the same AES-GCM algorithm and generates a new nonce that exactly matches the nonce that was generated by the client. The decryption process relies on extracting the ciphertext from the received packet, as on the client side:

data, *addr* = *server.recvfrom*(4096)

nonce = *HMAC*(*base_key*, *index*)

plaintext = *AESGCM*(*shared_key*).*decrypt*(*nonce*, *bytes.fromhex*(*ciphertext_hex*), *None*).*decode*()

After decryption, the router processes the request, re-encrypts the result using the same key and nonce, and then sends the encrypted response back to the client. This process ensures that the data is protected from attackers. It also completes the communication between the client and router, regardless of an attacker's presence on the network.

5.3 System Resource Consumption

Central Processing Unit (CPU) and Random Access Memory (RAM) usage were measured to evaluate the system's load under two conditions. The first is normal operation. The second is with an active attacker. The focus was on the Python process that runs the encryption and communication functions. Measurements were taken using Windows Task Manager on the client machine. This tool gives direct readings for each running process without affecting performance. Monitoring only the Python process allowed the system load to be separated from background activity.

For each condition, two readings were taken. The first was before starting encryption and transmission. The second was at peak activity during data exchange. This showed how much load the security functions add during operation. Measurements were repeated several times in the same virtual setup to keep conditions stable. The average values were plotted to compare CPU and RAM use between the two conditions. The data shows higher CPU and memory usage when an attacker is active. The extra load comes from encryption and verification. The increase stays within reasonable limits and does not reduce system stability. As shown Figure 7 and Figure 8.

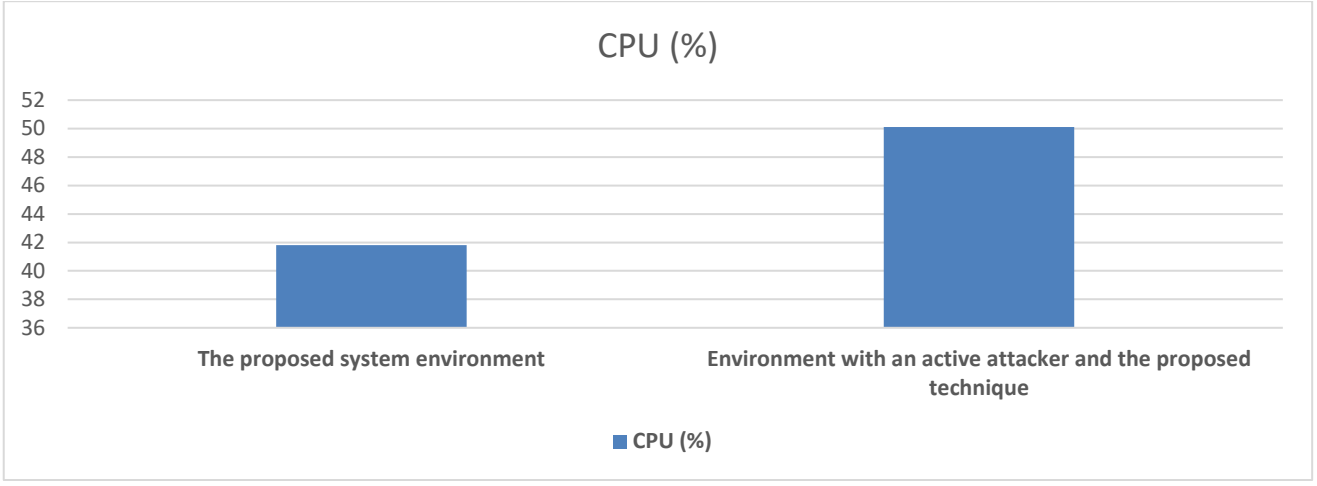

Figure 7: CPU use during application execution.

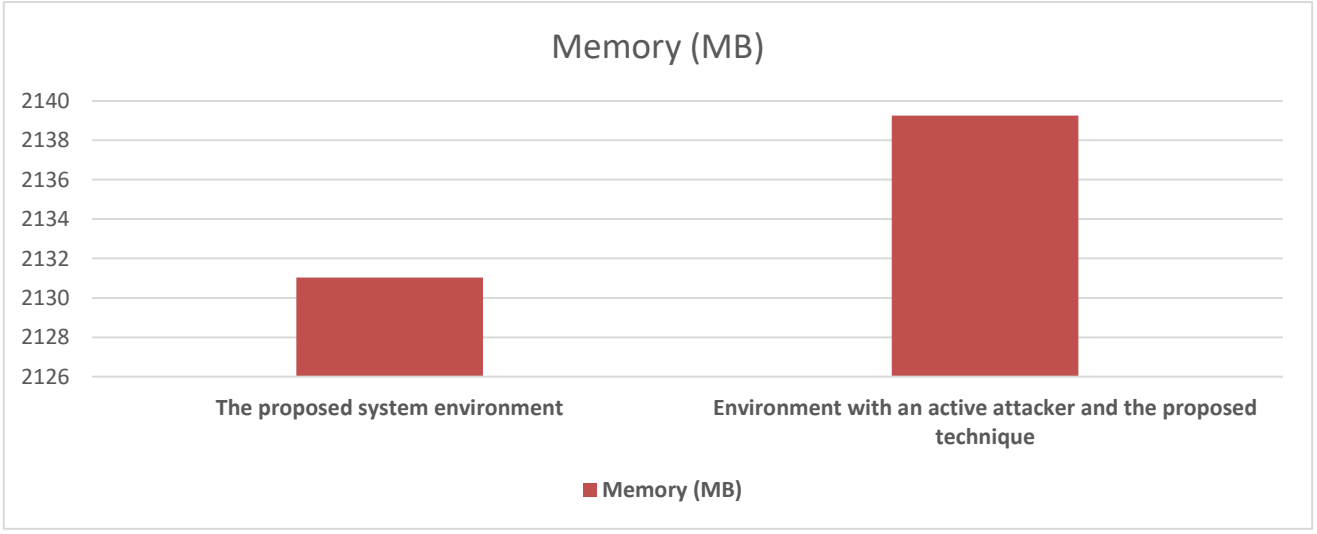

Figure 8: Memory use during program execution.

6. RESULTS

6.1 Execution

To assess application functional and security performance under many varying scenarios, this section presents the results of the systematic tests performed across four environments. These scenarios include the presence of an attacker, the absence of an attacker, and turning on a specialized encryption mechanism, and measuring two elements: Query/Response Integrity; and execution time. To guarantee measurement precision in each instance, all values were meticulously captured using Wireshark. The subsequent sub-sections illustrate the outcomes derived from the identical setting throughout the four situations.

6.1.1 Scenario 1 (ideal environment)

The first test evaluated the system's empirical performance in its basic form, in an ideal environment free of assailant or extra encryption. This test sought to track data flow between the client and the router during browser use. The following steps are followed during this testing:

Step 1: Set up the connection in VMware environment using a Host-Only network. Basically, the client and router created a limited local environment for complete data traffic management and guarantee total isolation from the internet.

Step 2: Run Wireshark on the client's network interface to track all incoming and exiting packets, especially with regard to HTTP and HTTPS protocols.

Step 3: In Google Chrome, input a search query: http://www.iana.org/domains/example.

Step 4: After sending the request, the packets are tracked using Wireshark.

Step 5: Using Wireshark's time analysis tools, the execution time and size are accurately calculated.

Step 6: Send the request (URLs) as plaintext using HTTP. During this process, the UDP protocol is deployed without any security layer assigned by the proposed system.

Step 7: At the end, all details are recorded for comparison with other cases. Table 3 shows the obtained results.

Table III: Results for ideal environment

Query		Response		Total time
Size	**Time**	**Size**	**Time**	
30	15.953573	81	16.009389	0.055816
30	22.981972	81	22.982611	0.000639
30	29.790260	81	29.790543	0.000283
Average	(0.055816 + 0.000639 + 0.000283) ÷ 3 =			0.0189127

Table 3 shows the results obtained when queries were run in a normal setting without any attacks or encryption. The first row had the longest execution time since the system was just starting up, which caused a little delay. The other rows show that replies are quick but not safe. This is because the adversaries could easily read all of the data since it is sent in clear format. Table 3 shows that the unsecured environment does not have any encryption or protection. It is therefore used as a baseline for comparing performance with the proposed solution later.

6.1.2 Scenario 2 (ideal environment with an active attacker)

This experiment is designed to measure the vulnerability of the client-router connection in the proposed system's absence when an active attacker is present within the local network. In this case, the attacker monitors the data flow between the client and the router while using the browser, enabling the attacker to view all user data. This is facilitated by the execution of the following procedures:

Step 1: Set up the environment by launching three virtual machines in a VMware Host-Only Network environment: First Computer (Client -Windows 10), second computer (the router -OpenWRT), and the third machine (Attacker with Kali Linux).

Step 2: Open the Kali Linux system and launch the attack tools. Specifically, the MITM tools described in the subsequent steps are activated.

Step 3: Launch ARP spoofing attack, whose goal is to corrupt the ARP table and divert client traffic to the attacker before sending it back to the router.

Step 4: In order to view plaintext content, SSL strip is used to intercept HTTPS requests and convert them to HTTP.

Step 5: Open Wireshark on the attacker's network interface to monitor all packets passing to and from the client. Thereafter, determine whether the data is readable or not.

Step 6: Submit a URL from the Google Chrome browser on the client device. This is accomplished by inputting the same search address as before, without the suggested system encryption enabled.

Step 7: Monitor the response. This involves tracking the data sent and received in Wireshark on the attacker's device, along with the packet content and clarity.

Step 8: Record the time the request was sent from the client's device, the response received, and the size of this request. Table 4 gives a depiction of the recorded data.

Table IV: Data on the presence of the attacker

Query		Response		Total time
Size	**Time**	**Size**	**Time**	
99	8.167677	108	8.231001	0.063324
99	12.340556	108	12.382725	0.042169
99	18.770788	108	18.813340	0.042552
Average	(0.063324 + 0.042169 + 0.042552) ÷ 3=			0.0493483

Table 4 represents an ideal environment in which an attacker is active. In this case, attack tools such as ARP spoof and SSL strip were activated in the Host-Only network and the attacker was able to intercept the connection and redirect the packets from HTTPS to HTTP. The findings indicated that the attacker is able to access the data in its entirety, which was transmitted as plaintext without any encryption. The overall response time surpassed that of Table 3. This is because the attacker was situated within the connection, causing a slight delay that did not hinder the penetration.

6.1.3 Scenario 3 (the proposed system environment)

This simulation aims to test the proposed system's safety and effectiveness when running in a normal operating environment (free of attackers), and evaluate its performance in the absence of threats. Here, it is necessary to isolate the system's impact on user experience and performance without any external interference. This scenario is accomplished through the execution of the following procedures:

Step 1: The environment is set, which comprises of the router (OpenWRT), the client device (Windows 10), and the Host-Only network in a VMware system.

Step 2: Run the proposed system between the client and the router. During this process, keys are transferred using the Diffie-Hellman algorithm. After the successful key exchange, a shared key is created and stored for usage in encryption.

Step 3: Apply the proposed system by inputting the search query: http://www.iana.org/domains/example in the custom browser created with PyQt5.

Step 4: After being text formatted, encrypt the connection using AES-GCM and the generated random nonce.

Step 5: Using JSON, capture the data and convert it into bytes. The data then travels to the router over UDP.

Step 6: The packet arrives at the router through the socket. Next, it is decoded using the same nonce and shared key.

Step 7: The response is encrypted the same way. Afterwards, it is forwarded to the client as the server response.

Step 8: The client receives the encrypted packet from the server. Next, it decodes it and presents the response in the Google Chrome browser.

Step 9: Using the internal system codes, the request time, response time, and full encryption time are logged. As shown in Table 5, Wireshark guarantees that the encrypted packets do not contain any readable data.

TABLE V: PACKET SIZE AND TIME IN THE PROPOSED SYSTEM ENVIRONMENT

Query		Response		Total time	Total encryption time
Size	**Time**	**Size**	**Time**		
259	2.015768	194	2.0268909	0.0111229	0.0221325
259	5.639932	194	5.682906	0.042974	0.042072
259	7.881923	194	7.915380	0.033457	0.037074
Average	(0.0221325 + 0.037074 + 0.042072) ÷ 3 =				**0.0337595**

Table 5 presents the results in the proposed system's environment, which is the full implementation scenario using AES-GCM encryption after Diffie-Hellman key exchange. The three rows indicate that all data was fully encrypted, with a separate encryption time for each operation. Encryption times were accurately measured, reaching 1.08 seconds in some cases. This demonstrates the significant performance burden occasioned by encryption. However, this encryption does not significantly impact the overall response time. As such, these results confirm the system's success in protecting data even in an ideal environment without an attacker.

6.1.4 Scenario 4 (an active attacker and the suggested methodology)

The execution of the encryption method on the same network amongst active attackers is the most critical situation. It serves as a practical assessment of the proposed system's efficacy. Specifically, this test aims to assess the system's ability to prevent data interception and analysis, even when advanced attack tools are utilized. The implementation procedures are as follows:

Step 1: Perform environment configuration. This comprises of the running (within a host-only network in a VMware setting comprising three virtual machines) the client (Windows 10), the router (OpenWRT), and the attacker (Kali Linux).

Step 2: Using the Diffie-Hellman protocol, generate the shared key between the client and the router. This is accomplished using a key exchange module.

Step 3: Perform custom browser configuration. The goal of this step is to encrypt each query utilizing a random nonce prior to exposing it to UDP and AES-GCM.

Step 4: Turn on Wireshark on the attacker to track all outgoing and incoming packets. Next, activate ARP spoof to poison the ARP table and intercept traffic. Thereafter, perform SSL strip whose goal is to convert HTTPS to HTTP. This enables the attackers to read the content of the exchanged messages.

Step 5: From the client side, a query is launched using the system's specialized browser to http://www.iana.org/domains/example, just as in the previous cases. In this setup, the encrypted link was contained in JSON. Therefore, the router decrypted the data and issued a re-encrypted answer.

It was noted that Wireshark only displayed indecipherable binary data. The SSL strip could not decode the data. This is because the protection is implemented at the application level, and not merely at the TLS level. As shown in Table 6, every bit that the adversary could see was indecipherable ciphertext.

Table VI: Size and time of the query and response

Query		Response		Total time	Total encryption time
Size	**Time**	**Size**	**Time**		
259	36.614364	194	36.626963	0.062599	0.074800
259	101.746798	194	101.813063	0.066265	0.075920
259	117.864350	194	117.876073	0.011723	0.067720
Average	(0.074800 + 0.075920 + 0.067720) ÷ 3 =				0.072813

The above results have illustrated the efficacy of the proposed solution in safeguarding communications within the local network from MITM stripping assaults. This has been facilitated by the analysis of four distinct operating scenarios described above. In the two unsecured setups, data was transferred in plaintext, either directly or after the attacker terminated HTTPS, rendering it susceptible to total eavesdropping. In contrast, the data exhibited a completely different behavior in the two operational cases: it was fully encrypted using AES-GCM, uninterpretable by surveillance tools, even with an active attacker. It was also observed that the system added an application-level security layer independent of HTTPS, preventing data leakage even when the TLS layer was compromised. Regarding performance, processing time increased from an average of 0.05 to 0.11 seconds, an acceptable difference in this operating environment. Regarding data volume, packet size increased significantly due to the inclusion of a nonce and encapsulation of the content in JSON, a normal cost of secure transmission. This highlights several advantages: physical protection against attacks, independent encryption, and easy integration with virtual infrastructures. The main setback is the limited increase in execution time and data volume. This calls for subsequent improvements in efficiency devoid of compromising safety. Figure 9 and Figure 10 give detailed depiction of the obtained results.

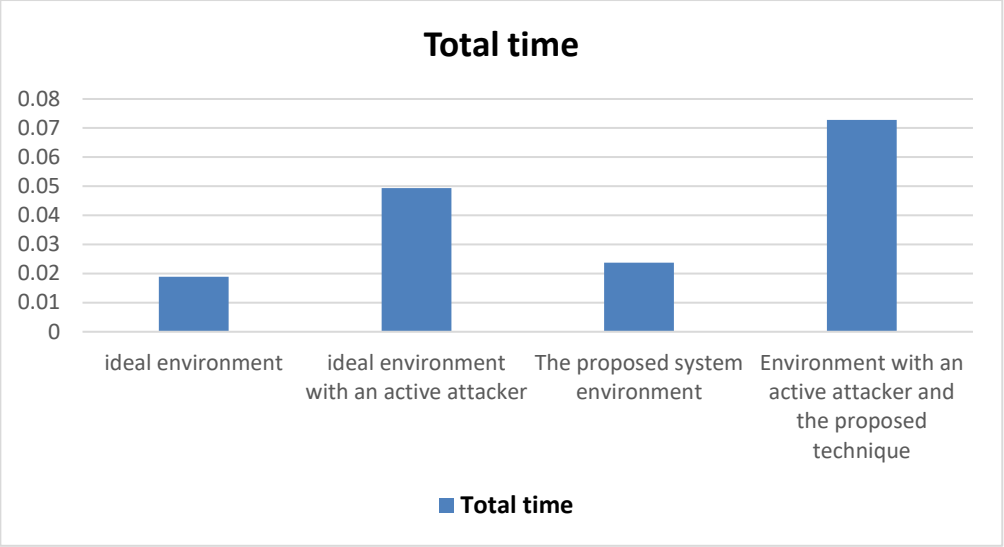

Figure 9: Total execution time comparison across all scenarios

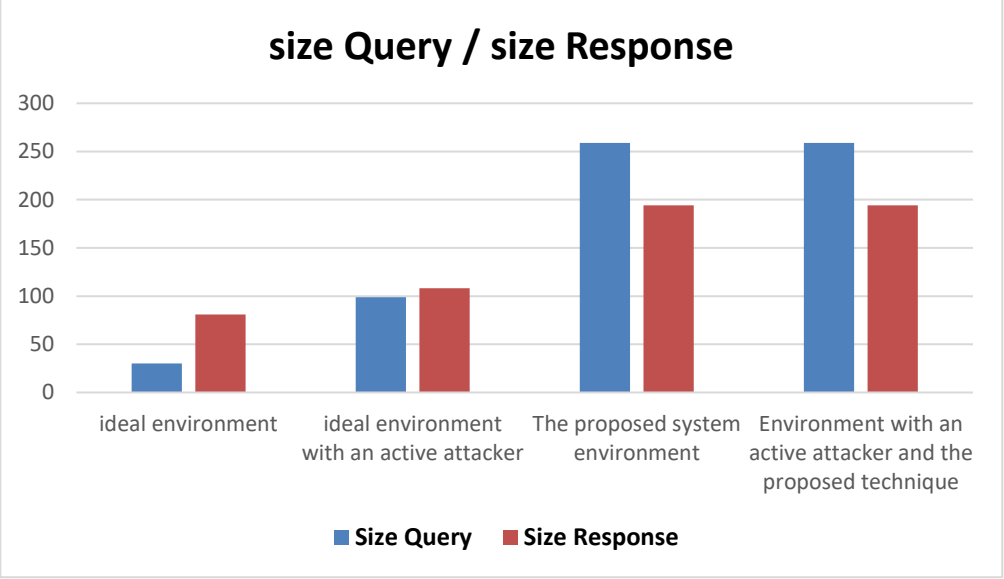

Figure 10: Query and response size comparison across all scenarios

7. SECURITY ANALYSIS

7.1 Formal Security Analysis

The Shamir, van de Riet, and Orponen (SVO) logic is a formal logic for analyzing authentication and security protocols. In this work, SVO is used instead of BAN or GYN because it offers support for public key infrastructure. The main network entities in the proposed method include Client (C), Router (R) and DNS Server (S).

7.1.1 Principals, beliefs and statements

- M: Message or value
- P, Q: Principals (such as C, R, S)
- $P \mid\equiv M$: Principal P believes (or is entitled to believe) M
- $P \ni M$: P possesses (or is capable of possessing) M
- $\#(M)$: M is fresh (not used before in any other run)
- $P \overset{K}{\leftrightarrow} Q$: P and Q may use the shared key K (good key)
- $P \Rightarrow M$: P has jurisdiction over M (controls its validity).
- $P \lhd M$: P sees M (is sent M in a message)
- $P \mid\sim M$: P once said M

- $PK(P, K)$: K is P's public key.
- $\{M\}_K$: M is encrypted/decrypted with key K.
- (X, Y): Concatenation of X and Y
- $X \subset Y$: X is a sub term of Y (such as nonce inside a message).

7.1.2 SVO Rules

Rule 1: Message Meaning Rule (MMR) (Shared Key)

If P believes K is shared with Q and sees $\{X\}K$, then P believes Q said X. Mathematically, this is expressed as follows:

$P \models (P \leftrightarrow Q: K), P \ni \{X\}K$

$P \models Q \mid\sim X$

Rule 2: Freshness Propagation Rule (FPR)

If part of a message is fresh, the entire message is considered fresh. Mathematically, this is written as follows:

$\#(X), X \subset Y$

$\#(Y)$

Rule 3: Jurisdiction Rule (JR)

If P trusts Q on X and Q believes X, then P believes X. This is mathematically expressed as shown below.

$P \models (Q \Rightarrow X), P \models Q \models X$

$P \models X$

7.1.3 Assumptions

7.1.3.1 Initial possessions

$C \ni g, p$ (Client C possesses generator g and prime p)
$R \ni g, p$ (Router R possesses generator g and prime p)

7.1.3.2 Freshness

$C \models \#(a)$ (Client C believes a is fresh)
$R \models \#(b)$ (Router R believes b is fresh)
$C \models R \models \#(n)$ (Client C and Router R believe n is fresh)

7.1.3 Computation Ability

$C \ni A = g\text{^}a \bmod p$
$R \ni B = g\text{^}b \bmod p$

7.1.4 Protocol Steps

1. $C \lhd A$: Client sent A
2. $R \lhd A$: Router sees A
3. $R \lhd B$: Router sent B
4. $C \lhd B$: Router sees B
5. $C \ni K = B\text{^}a \bmod p$
6. $R \ni K = A\text{^}b \bmod p$
7. $C \ni \#(n)$, $\{\text{URL}\}_{K,n}$: encryption URL
8. $\{\text{URL}\}_K \subset DQ$: encrypted URL is a sub term of DQ
9. $C \lhd DQ$: Client sent DQ
10. $R \lhd DQ$: Router sees DQ
11. $R \ni \#(n)$, $\{URL, QID\}_{K,n}$: decryption URL
12. $R \lhd DQ$: Router sent DQ
13. $S \lhd DQ$: DNS Server sees DQ
14. $S \lhd DR$: DNS Server sent DR
15. $R \lhd DR$: Router sees DR
16. $R \ni \#(n)$, $\{DR, QID\}_{K,n}$: encryption URL
17. $R \lhd DR$: Router sent DR
18. $C \lhd DR$: Client sees DR
19. $C \ni \#(n)$, $\{URL, QID\}_{K,n}$: decryption URL
20. $C \lhd HTTP\ req$: Client sent HTTP request
21. $WS \lhd HTTP\ req$: Web Server sees HTTP request

22. $WS \triangleleft HTTP\ resp$: Web Server sent HTTP response
23. $C \triangleleft HTTP\ resp$: Client sees HTTP response

7.1.5 Derive Beliefs

According to Steps 1-6, Rule 1, Rule 2 and the initial assumptions, the Client C and Router R share a secret key and some fresh key. This key is only used between C and R, that is:

$$C \overset{K}{\leftrightarrow} R;\ C \mid\equiv R \mid\equiv \#(K)$$

Based on Steps 7-12 and Rule 3, the Router R beliefs that the encrypted URL came from the client C. In addition, both parties belief n is fresh value, that is:

$$R \mid\equiv (C \Rightarrow \text{URL});\ C \mid\equiv R \mid\equiv \#(n)$$

According to Steps 12-15, secure messages exchanges can take place between DNS server and Router R.

Based on Steps 16-19 and Rule 3, Client C beliefs that encrypted URL came from Router R as a response to the DNS query. Therefore,

$$C \mid\equiv (R \Rightarrow \text{URL});\ R \mid\equiv C \mid\equiv \#(n)$$

According to Steps 20-23, secure message exchanges occur between web server and Client C.

7.1.6 Summary

- Both parties agree on the same session key.
- The key is fresh, that is, $(\#(K))$.
- The key is known only to Client C and Router R (key secrecy).
- The key is suitable for secure communication between Client C and Router R.
- The symmetric encryption has fresh $(\#(n))$ for each direction.

7.2 Informal security analysis

This section examines the proposed system's security model that mitigates MITM stripping attacks. In addition, we evaluate its efficacy against diverse security threats.

7.2.1 Security modeling

The proposed system is dependent on three primary components, whose details are described below:

- **Victim:** the user's device that establishes connections through the browser.
- **Attacker:** an intermediary that seeks to disrupt and convert connections from HTTPS to HTTP or alter keys during the exchange process.
- **Enhanced router:** incorporates an intelligent analysis module to authenticate the key's validity and avert stripping.

The proposed system comprises of the following functions:

- Key generation uses the Diffie-Hellman method, with the key briefly retained in a local file at both endpoints.
- The client encrypts the message using AES-GCM and transmits it to the router through the designated port (1600).
- The router decrypts messages utilizing the shared key while validating GCM integrity.
- The system prevents unencrypted connections or those utilizing altered keys.

The system employs a session-specific key that is generated singularly and automatically altered for each new connection attempt. Therefore, the proposed system prevents the reuse of keys or messages.

7.2.2 Security Proofs

The fundamental principle of the proposed system, which is primarily tailored for MITM stripping attacks prevention, is that for the router to produce a valid response decrypted and authenticated by the client, the message must be encrypted with the appropriate shared key and remain unchanged during transmission.

Theorem 1: *an attacker cannot produce a legitimate encryption response without the shared key.*

Proofs 1: In the proposed system, router decryption depends on the GCM tag aligning with the message data. Any alteration of the data or use of an alternative key leads to an instantaneous verification failure, eliminating the necessity for human examination.

Theorem 2: *a perpetrator cannot effectively retransmit identical messages (replay attack).*

Proofs 2: Each key exchange session generates a distinct key. Even if an encrypted communication is intercepted, an attacker cannot exploit it because of the unique shared key for each session, which is purged from memory upon connection termination.

Theorem 3: *an assailant cannot execute a stripping attack to transition the connection from HTTPS to HTTP.*

Proofs 3: The proposed system only accepts encrypted requests with the shared key and valid GCM tag. Any effort to convert the connection to HTTP yields an unintelligible message that is promptly rejected by the router.

Theorem 4: *The developed system facilitates effective session-oriented key management.*

Proofs 4: In our scheme, each key is produced during a singular session and retained in a temporary file that is systematically eradicated upon session termination. This key is not permanently stored. Instead, it is restricted to the session's lifetime.

Theorem 5: *The system preserves confidentiality and inhibits content revelation.*

Proof 5: In the proposed system, we employ AES-GCM with a shared key to ensure that adversaries are unable to access or alter the information devoid of the required key. During decryption, changing any byte results in GCM failure.

Theorem 6: *The system effectively mitigates impersonation and key alteration attacks*.

Proof 6: An attacker cannot alter messages or produce a key identical to the client without possessing the confidential parameters of the Diffie-Hellman algorithm (private keys). This is mathematically infeasible due to the difficulties of solving the DHP.

8. CONCLUSIONS

This study examined the risk of MITM Stripping attacks and developed an application-level protection mechanism that integrates authenticated encryption with secure key exchange. A virtual environment was created to replicate real-world attack scenarios. The results confirmed that the system prevents HTTPS downgrades, protects data confidentiality, and prevents attackers from reading or modifying communications, even when they are within the same local network. Both formal and informal analyses demonstrated the system's robustness.

The proposed system benefits from three aspects: First, it creates an independent application-level security layer that continues to protect data even in the event of a breach. Furthermore, it seamlessly integrates into virtual testbeds and router-based infrastructures, facilitating experimentation across diverse contexts. The performance overhead regarding execution time and packet size was satisfactory, demonstrating the design's viability. This work may prove beneficial in situations where HTTPS security is inadequate or breached. The design illustrates that fundamental encryption aspects, when integrated with astute traffic management, can bolster the robustness of communication systems against active threats. The studies confirm that the method developed is effective against MITM stripping attacks, while also allowing for considerable avenues for improvement and research. Next steps for the work include:

- Using lightweight encryption algorithms, such as Ascon or PRESENT, may lessen the computational costs related to the technique, allowing for wider deployment towards resource-constrained devices like IoT nodes.
- Improving code error handling that will allow the index counter to be automatically rolled forward in case of a decryption/message verification failure to improve session stability and support session continuity.
- Expand the model to investigate mobile environments and wireless technologies, where MITM attacks are on the rise due to the use of public Wi-Fi networks.
- Investigate hybrid approaches, combining machine learning for on-the-fly detection of anomalous traffic patterns, along with adding an encryption-based security framework.

9. LIMITATIONS

The system has not been fully tested in a real-world environment. The application was conducted in a virtual environment that simulates reality but does not represent all real-world network conditions. Real networks vary in terms of router types, endpoint types, network configurations, and workloads. These factors may lead to unexpected problems during operational use.

The system relies on key exchange and accurate nonce value generation at each stage of the communication. Any error in the messages or nonce calculation results in an immediate failure of the authentication process and failure of data transfer. This represents one of the current technical limitations of the system.

Conflicts of Interest

The authors declare no conflicts of interest.

Funding

None.

Acknowledgment

None.

References

[1] O. A. Beg, A. A. Khan, W. U. Rehman, and A. Hassan, "A review of AI-based cyber-attack detection and mitigation in microgrids," *Energies*, vol. 16, no. 22, pp. 1–23, 2023, doi: 10.3390/en16227644.

[2] D. S. Zhraw, M. A. Hussain, Z. A. Abduljabbar, and V. O. Nyangaresi, "Chronological review of MITM attacks: challenges, solutions, and recommendations," in Cybersecurity Threats, Actors, and Dynamic Mitigation, Springer, 2025, pp. 1–20, doi: 10.1007/978-3-032-03406-9.

[3] M. A. Hussain, Z. A. Abduljabbar, H. F. Neamah, and A. H. Ali, "Provably throttling SQLI using an enciphering query and secure matching," *Egyptian Informatics Journal*, vol. 23, no. 4, pp. 145–162, 2022, doi: 10.1016/j.eij.2022.10.001.

[4] Z. A. Abduljabbar, H. Jin, Z. A. Hussien, and A. A. Yassin, "Towards Efficient Authentication Scheme with Biometric Key Management in Cloud Environment," in Proc. IEEE 2nd Int. Conf. Big Data Security on Cloud Computing, 2016, pp. xx–yy, doi: 10.1109/BigDataSecurity-HPSC-IDS.2016.25.
[5] Z. A. Abduljabbar, V. O. Nyangaresi, H. M. Jasim, J. Ma, M. A. Hussain, Z. A. Hussien, and A. J. Y. Aldarwish, "Elliptic Curve Cryptography-Based Scheme for Secure Signaling and Data Exchanges in Precision Agriculture," Sustainability, vol. 15, no. 13, art. 10264, 2023, doi: 10.3390/su151310264.
[6] F. Ahmad, F. Kurugollu, A. Adnane, R. Hussain, and F. Hussain, "MARINE: Man-in-the-Middle Attack-Resistant Trust Model in Connected Vehicles," IEEE Internet of Things J., vol. 7, no. 5, pp. 4662–4674, 2020, doi: 10.1109/JIOT.2020.2967568.
[7] F. Algarni, M. A. Khan, W. Alawad, and N. Ben Halima, "P3S: Pertinent Privacy-Preserving Scheme for Remotely Sensed Environmental Data in Smart Cities," IEEE J. Sel. Topics Appl. Earth Obs. Remote Sens., vol. 16, pp. 5905–5918, 2023, doi: 10.1109/JSTARS.2023.3288743.
[8] M. A. Al-shareeda, M. Anbar, S. Manickam, and I. H. Hasbullah, "Review of Prevention Schemes for Man-In-The-Middle (MITM) Attack in Vehicular Ad hoc Networks," Int. J. Eng. Manag. Res., vol. 10, no. 3, pp. 153–158, 2020, doi: 10.31033/ijemr.10.3.23.
[9] H. El-Taj and L. Miralam, "Network Sniffing and Its Consequences: A Comprehensive Survey," Int. J. Comput. Sci. Inf. Secur. (IJCSIS), vol. 22, no. 3, pp. 1–7, 2024, doi: 10.5281/zenodo.12750103.
[10] M. Ozkan-Okay, A. A. Yilmaz, E. Akin, A. Aslan, and S. S. Aktug, "A Comprehensive Review of Cyber Security Vulnerabilities," Electronics, vol. 12, no. 6, p. 1333, 2023, doi: 10.3390/electronics12061333.
[11] D. R. L. Brown and R. P. Gallant, "The Static Diffie-Hellman Problem," IACR Cryptol. ePrint Arch., Rep. 2004/306, pp. 1–17, 2004. [Online]. Available: https://eprint.iacr.org/2004/306.
[12] V. O. Nyangaresi, Z. A. Abduljabbar, H. M. Jasim, and M. A. Hussain, "Smart City Energy Efficient Data Privacy Preservation Protocol Based on Biometrics and Fuzzy Commitment Scheme," Sci. Rep., pp. 1–17, 2024, doi: 10.1038/s41598-024-67064-z.
[13] C. Gupta and N. V. S. Reddy, "Enhancement of Security of Diffie-Hellman Key Exchange Protocol Using RSA Cryptography," J. Phys. Conf. Ser., vol. 2161, no. 1, art. 012014, 2022, doi: 10.1088/1742-6596/2161/1/012014.
[14] S. Anand and V. Perumal, "EECDH to Prevent MITM Attack in Cloud Computing," Digit. Commun. Networks, vol. 5, no. 4, pp. 276–287, 2019, doi: 10.1016/j.dcan.2019.10.007.
[15] A. M. Abdullah, "Advanced Encryption Standard (AES) Algorithm to Encrypt and Decrypt Data," Cryptogr. Netw. Secur., vol. 16, no. 1, p. 11, 2017.
[16] I. A. Awan, M. Shiraz, M. U. Hashmi, Q. Shaheen, R. Akhtar, and A. Ditta, "Secure Framework Enhancing AES Algorithm in Cloud Computing," Secur. Commun. Networks, vol. 2020, 2020, doi: 10.1155/2020/8863345.
[17] M. S. Hossain, A. Paul, M. H. Islam, and M. Atiquzzaman, "Survey of the Protection Mechanisms to the SSL-Based Session Hijacking Attacks," Netw. Protoc. Algorithms, vol. 10, no. 1, p. 83, 2018, doi: 10.5296/npa.v10i1.12478.
[18] A. R. Chordiya, S. Majumder, and A. Y. Javaid, "Man-in-the-Middle (MITM) Attack Based Hijacking of HTTP Traffic Using Open Source Tools," in Proc. IEEE Int. Conf. Electro Inf. Technol., May 2018, pp. 438–443, doi: 10.1109/EIT.2018.8500144.
[19] M. A. Jonas, M. S. Hossain, R. Islam, H. S. Narman, and M. Atiquzzaman, "An Intelligent System for Preventing SSL Stripping-Based Session Hijacking Attacks," in Proc. IEEE Mil. Commun. Conf. (MILCOM), Nov. 2019, pp. 1–6, doi: 10.1109/MILCOM47813.2019.9021026.
[20] S. Duddu, A. R. Sai, C. L. S. Sowjanya, G. Ramakoteswara Rao, and K. S. Siddabattula, "Secure Socket Layer Stripping Attack Using Address Resolution Protocol Spoofing," in Proc. Int. Conf. Intell. Comput. Control Syst. (ICICCS), 2020, pp. 973–978, doi: 10.1109/ICICCS48265.2020.9120993.
[21] H. A. S. Adjei, M. T. Shunhua, G. K. Agordzo, Y. Li, G. Peprah, and E. S. A. Gyarteng, "SSL Stripping Technique (DHCP Snooping and ARP Spoofing Inspection)," in Proc. Int. Conf. Adv. Commun. Technol. (ICACT), Feb. 2021, pp. 187–193, doi: 10.23919/ICACT51234.2021.9370460.
[22] V. Kampourakis, G. Kambourakis, E. Chatzoglou, and C. Zaroliagis, "Revisiting Man-in-the-Middle Attacks Against HTTPS," Netw. Secur., vol. 2022, no. 3, 2022, doi: 10.12968/S1353-4858(22)70028-1.
[23] M. S. Ali, S. Salisu, U. Y. Magaji, and A. John, "An Empirical Study of Mobile Apps SSL Stripping Man-in-the-Middle Attack," Niger. J. Comput. Eng. Technol. (NIJOCET), vol. 1, no. 1, pp. 77–87, Jun. 2022, doi: 10.4225/75/57b65a25343ce.
[24] Z. Cekerevac, P. Cekerevac, L. Prigoda, and F. Al-Naima, "Security Risks from the Modern Man-in-the-Middle Attacks," MEST J., vol. 13, no. 13, pp. 1–10, Jan. 2025, doi: 10.12709/mest.13.13.01.04.
[25] P. K. Swain, S. Satpathy, and S. R. Pokuri, "Machine Learning Based Strategy for MITM Attack Mitigation Utilizing Hybrid Feature Selection," Proc. Eng. Sci., vol. 7, no. 1, pp. 615–624, 2025, doi: 10.24874/PES07.01D.017.
[26] K. F. Tang, C. W. Tu, S. L. A. Mak, and S. Y. Chau, "A Multifaceted Study on the Use of TLS and Auto-Detect in Email Ecosystems," in Proc. NDSS Symp., 2025.
[27] H. Fereidouni, O. Fadeitcheva, and M. Zalai, "IoT and Man-in-the-Middle Attacks," Security and Privacy, vol. 8, no. 2, 2025, doi: 10.1002/spy2.70016.